Formal Verification of Model Refactorings for Hybrid Control Systems

vorgelegt von
Diplom-Informatiker
Sebastian Schlesinger
geb. in Dresden

von der Fakultät IV – Elektrotechnik und Informatik
der Technischen Universität Berlin
zur Erlangung des akademischen Grades

Doktor der Ingenieurwissenschaften
– Dr.-Ing. –

genehmigte Dissertation

Promotionsausschuss:

Vorsitzender:	Prof. Dr. Andreas Vogelsang
Gutachterin:	Prof. Dr. Sabine Glesner
Gutachter:	Prof. Dr. Holger Giese
Gutachterin:	Prof. Dr.-Ing. Ina Schäfer

Tag der wissenschaftlichen Aussprache: 19. Juli 2018

Berlin 2018

Abstract

The ever growing complexity in modern embedded systems require to incorporate increasingly many functions into a single system. Such increasing functionality leads to growing design complexity. Model Driven Engineering (MDE) has been proposed to improve the complexity management for development of embedded systems. An industrially widely used technique to reduce the complexity of models and establish compliance with industrial MDE guidelines are refactorings. However, refactorings are often performed manually and may introduce erroneous behaviour. This may be critical especially in safety-critical environments. Thus, techniques to formally establish behavioural equivalence of source and target model of a refactoring are highly desirable.

The main challenge for defining an approach for formal verification of behavioural equivalence of embedded systems is that these systems typically are hybrid control systems in the sense that they comprise both time-discrete and time-continuous behaviour. As a consequence, the data flow-oriented languages that are used to model such systems inherently have an approximate nature. This means that even though two models may be equivalent if interpreted in an ideal mathematical domain, their evaluation by a semantics of a data flow-oriented language may yield different values due to numerical approximations.

In this thesis, we present an approach for the semi-automatic formal verification of behavioural equivalence of hybrid control systems modelled in data flow-oriented languages. Our approach is formally well-founded, addresses the approximate nature of data flow-oriented languages, is automatable, and scales comparatively well for many industrially relevant applications. Our major contribution is twofold: Firstly, we provide a sound methodology for the formal verification of behavioural equivalence of hybrid control models on data flow-oriented languages. Secondly, we largely automate our methodology for a subset of such models. We select Simulink as major representative of industrially relevant data flow-oriented languages. Our theoretical methodology is based on a formal operational semantics for Simulink, which we have adopted and extended to fit our needs. Based on the operational semantics, we define a novel denotational semantics, our so-called Abstract Representation, which describes an idealised behaviour of Simulink in terms of differential and difference equations. We show the soundness of our Abstract Representation with respect to the operational semantics. Then, we enable the formal verification of behavioural equivalence of semantical refactorings. An example for such a semantical refactoring is the replacement of an explicitly modelled differential

equation by its analytical solution. To address the approximate nature of data flow-oriented languages, we adapt the equivalence notion of approximate bisimulation to Simulink. States are approximately bisimilar with a precision $\varepsilon \geq 0$ if the distance of their values is at most ε. We provide sufficient conditions to conclude approximate bisimulation for two given models executed in the same finite simulation interval $\mathcal{I}$, and we provide means to calculate the precision $\varepsilon(\#\mathcal{I})$, which depends on the length of the simulation interval. Note that this differs from the usual application of approximate bisimulation in the literature. There, approximate bisimulation is applied to transition systems with infinite runtimes, i.e., the precision ε holds for arbitrary lengths of execution intervals.

We have largely automated our approach for the subset of models expressible as linear differential or difference equations. Via Laplace transform and z-transform, we are able to provide an almost fully automated, modular verification approach that uses a Computer Algebra System to perform the equivalence checks and calculations. We have implemented our automation approach and demonstrated its performance with experimental results.

Zusammenfassung

Die wachsende Komplexität moderner eingebetteter Systeme erfordert es, immer mehr Funktionalität in die Systeme zu implementieren. Solch wachsende Funktionalität führt zu wachsender Komplexität im Design. Modellgetriebene Entwicklung (MDE) ist ein Mittel, um das Management der Komplexität zu verbessern. Eine industriell weit verbreitete Technik zur Reduktion der Modellkomplexität und zur Herstellung der Konformität der Modelle mit industriellen MDE Richtlinien sind Refactorings. Refactorings werden jedoch oft manuell durchgeführt und können daher oft fehlerhaft sein. Dies kann kritisch sein, insbesondere in sicherheitskritischen Umgebungen. Techniken, die es erlauben, die Verhaltensäquivalenz von Quell- und Zielmodell von Refactorings formal zu verifizieren sind daher wünschenswert.

Eine besondere Herausforderung bei der Definition einer Methodik für die formale Verifikation von Verhaltensäquivalenz für eingebettete Systeme ist, dass diese Systeme typischerweise hybride Kontrollsysteme sind, d.h. sie beinhalten zeit-diskretes und zeit-kontinuierliches Verhalten. Aus diesem Grunde weisen datenflussorientierte Sprachen, die zur Modellierung solcher Systeme verwendet werden, eine approximative Natur auf. Damit ist gemeint, dass obwohl zwei Modelle, interpretiert in einer idealisierten, mathematischen Weise, äquivalent sein mögen, es vorkommen kann, dass die Auswertung dieser Modelle durch eine Semantik einer datenflussorientierten Sprache zu abweichenden Werten durch numerische Approximationen führt.

In dieser Dissertation präsentieren wir einen Ansatz für die semi-automatische formale Verifikation der Verhaltensäquivalenz von hybriden Kontrollsystemen, die in einer datenflussorientierten Sprache modelliert sind. Unser Ansatz ist formal fundiert, berücksichtigt die approximative Natur der datenflussorientierten Sprachen, ist automatisierbar und skaliert relativ gut für viele industriell relevante Anwendungen. Der Ansatz besteht aus zwei Hauptteilen. Zunächst definieren wir eine theoretische Methodik für die formale Verifikation der Verhaltensäquivalenz von hybriden Kontrollmodellen auf datenflussorientierten Sprachen. Danach automatisieren wir weitgehend die Methodik für eine Teilmenge dieser Modelle. Wir wählen Simulink als Hauptvertreter der industriell relevanten datenflussorientierten Sprachen. Unsere theoretische Methodik basiert auf einer formalen operationalen Semantik für Simulink, die wir angepasst und erweitert haben, um sie für unsere Zwecke nutzbar zu machen. Wir beginnen mit der Definition einer denotationellen Semantik, unserer sogenannten Abstract Representation, die ein idealisiertes Verhalten von Simulink mittels Differential- und Differenzengleichungen beschreibt. Wir zeigen die Korrektheit unserer Abstract Representation in Bezug auf die operationale Semantik. Die Abstract Representation ermöglicht die Verifikation der

Verhaltensäquivalenz bei semantischen Refactorings, z.B. der Ersetzung eines Modells, das durch eine Differentialgleichung repräsentiert wird durch ein Modell, das deren analytische Lösung repräsentiert. Um die approximative Natur der datenflussorientierten Sprachen zu berücksichtigen, passen wir den Äquivalenzbegriff der approximierten Bisimulation an Simulink an. Zustände sind approximativ bisimilar mit einer Präzision $\varepsilon \geq 0$ wenn ihre Werte höchstens ε voneinander entfernt sind. Wir präsentieren anschließend hinreichende Bedingungen, um auf die approximierte Bisimulation für zwei gegebene Modelle, die im selben endlichen Simulationsintervall $\mathcal{I}$ ausgeführt werden, schließen zu können. Dies beinhaltet Methoden zur Berechnung der Präzision $\varepsilon(\#\mathcal{I})$, die von der Länge des Simulationsintervalls abhängt. Beachten Sie, dass dies von der in der Literatur üblichen Anwendung der approximierten Bisimulation abweicht. In der Literatur wird approximierte Bisimulation üblicherweise auf Transitionssysteme mit unendlicher Laufzeit angewandt, d.h. die Präzision ε gilt für beliebig lange Zeiten der Ausführung.

Wir haben unseren Ansatz für eine Teilmenge von Modellen, die als lineare Differential- oder Differenzengleichungen darstellbar sind, weitgehend automatisiert. Mit Hilfe der Laplace- und z-Transformation sind wir in der Lage, eine nahezu vollständig automatisierte, modulare Lösung zur Verifikation zu präsentieren. Sie verwendet ein Computer Algebra System für die Äquivalenzcheckes und zur Berechnung. Wir haben unseren Automatisierungsansatz implementiert und dessen Leistung mit experimentellen Ergebnissen demonstriert.

Danksagung

Die folgende Arbeit entstand während meiner Tätigkeit als wissenschaftlicher Mitarbeiter im Fachgebiet „Software and Embedded Systems Engineering“ unter der Leitung von Prof. Dr. Sabine Glesner. Bei ihr möchte ich mich herzlich bedanken. Sie hat immer an mich geglaubt und meine Finanzierung auch in schwierigen Zeiten durchgesetzt. Ohne sie wäre die Arbeit nicht zustande gekommen.

Darüber hinaus möchte ich mich bei meinen beiden Kollegen und Freunden Dr.-Ing. Paula Herber und Dr.-Ing. Thomas Göthel sehr herzlich bedanken. Ich danke Euch sehr für all die inhaltlichen Diskussionen, wiederholtes Korrekturlesen und dass ihr immer für mich da wart. Ihr seid echt sehr gute Freunde! Ganz herzlich danke ich auch meinem Kollegen und Freund Marcus Mikulcak. Mit ihm hatte ich stets jemanden, mit dem ich mich jederzeit sehr gut verstand und der mir auch mental schwierige Zeiten zu überstehen half. Allen meinen Kollegen möchte ich danken für viele hilfreiche Diskussionen und für die sehr angenehme Atmosphäre in unserer Gruppe. Wir sind alle von Kollegen zu Freunden geworden.

Zuletzt möchte ich meiner Familie und all meinen Freunden für ihre Unterstützung danken, insbesondere meiner Frau Silva und meinen Eltern Gabriele Schlesinger und Matthias Großmann.

Contents

Chapter 1

Introduction

Embedded systems are ubiquitous. They are employed in various domains, e.g. avionics, automotive, consumer electronics, telecommunications, and healthcare technology. Embedded systems carry out observation and control functions. They combine time-continuous and time-discrete parts, which is referred to as *hybrid* behaviour. The development of hybrid control systems often follows a Model Driven Engineering (MDE) approach. In MDE, executable models are the focal entities being developed in the process. Typical attributes of embedded systems such as hybrid behaviour, obtaining environmental data, calculating a response to adapt the behaviour, and feed it back to the environment via actuators, call for signal flow-oriented or data flow-oriented languages to support the development process. Examples are MATLAB Simulink [TM17b] and Modelica [Ass17]. The time-continuous part in such models introduces an approximation error caused by the numerical execution on machines with finite execution time and storage space. Mathematically, the time-continuous parts form a continuous function, i.e., values are assigned to uncountably many time steps. However, the execution is performed in finite steps. This implies that during the execution, the values are approximated.

Software development for embedded systems is increasingly complex: According to [BBH+14], a typical upper class vehicle in 2007 contained about 270 implemented functions to interact with its driver. Upper class vehicles in 2014, the year of the article, contain more than 500 of such functions. *Model refactoring* is a common technique to cope with the ever-increasing complexity in nowadays' applications and to establish compliance with industrial MDE guidelines via model transformations, e.g. as defined in ISO 26262 [Int11] for safety-critical systems in the automotive domain. However, such model transformations are often performed manually or by utilising unverified tools. They therefore bear the risk of introducing unwanted or even erroneous behaviour in the process. Especially in safety-critical environments, it is crucial to rigorously ensure *correct* refactorings, i.e., that source and target model are semantically equivalent. Currently, there exists no approach

for proving behavioural equivalence of systems modelled in data flow-oriented languages for hybrid control systems.

In this dissertation, we aim for a semi-automated approach to verify the behavioural equivalence of hybrid control systems modelled in a data flow-oriented language and their refactored counterpart.

We require the following criteria to be met by our approach.

1. Formal Foundation: Since this thesis aims for a formal verification approach, we require an unambiguous, precise semantics.

2. Approximate Nature: We aim at supporting the approximate nature of data flow-oriented modelling languages for hybrid control systems, which is introduced by the numerical execution.

3. Behavioural Equivalence: We require a suitable and concise notion for behavioural equivalence.

4. Automation: Manual verification is time-consuming and error prone. An at least semi-automated solution is therefore desired.

5. Scalability: We aim for an approach capable of verifying behavioural equivalence on models of industrially relevant sizes.

In the following, we describe the key ideas of our proposed solution.

We select Simulink as major representative of an industrial modelling language for hybrid control systems. The reasons are 1) Simulink has a broad acceptance, it is widely used across many industries, e.g. automotive, aerospace industry, and 2) it exhibits typical properties of modelling languages used in the industrial context for MDE of embedded hybrid control systems, such as data-flow orientation, numerical simulation, use of various arithmetic, stateful, and control flow blocks. The former reason strengthens the claim for industrial applicability, while the latter reason implies that the concepts of this thesis can be transferred to other data flow-oriented languages that are used for hybrid control systems.

With our approach, we provide 1) a well-founded methodology that enables formal verification of behavioural equivalence of Simulink models, and 2) an automation of our approach for behavioural equivalence checking of a subset of hybrid control systems modelled in Simulink. The subset we support are basically models whose behaviour can be expressed by *linear* differential or difference equations. We perform the automated verification of behavioural equivalence with the help of a Computer Algebra System.

Our main contributions are:

1. *Denotational Semantics* We define an *Abstract Representation* (AR) that describes the behaviour of Simulink models in terms of differential and difference equations. The function that solves the differential or difference equations provides us with a well-defined denotational semantics. Our denotational semantics is sound with the formal operational semantics [BC12] for MATLAB/ Simulink, which we use as our formal basis. The reason we introduce our AR and use it for verification of behavioural equivalence is that we compare whole trajectories rather than steps in the operational semantics. Trajectories are solutions of the AR's equations. They describe an idealised behaviour of the models. This in turn enables use of symbolic approaches via Computer Algebra Systems. Moreover, it enables us to support refactorings expressing analytical solutions of underlying differential or difference equations. We show the soundness of our AR and denotational semantics with the operational semantics [BC12] for Simulink in the following sense: Executions by the operational semantics with decreasing simulation step size yields a sequence of trajectories that uniformly converge against the solutions induced by our AR.

2. *Equivalence Notion* We adapt the concept of *approximate bisimulation* (introduced e.g. in [GP11]) to obtain a suitable, well-established formal equivalence notion that properly addresses the approximate nature of the Simulink semantics. The approximate nature means that although two models interpreted as dynamical systems in our denotational semantics may represent the same solution from a mathematical perspective, the operational semantics may yield differing values. Approximate bisimulation introduces a precision ε, i.e., a maximal distance or ε tube in which the values of the executions of source and target model may reside. This ε generally increases with the length of the interval of the execution of the Simulink models. Note that this differs from the application of approximate bisimulation as given in [GP11]. There, approximate bisimulation is applied to transition systems with infinite runtimes, i.e., the precision ε holds for arbitrary lengths of execution intervals.

3. *Proof Scheme* We provide a proof scheme, i.e., a methodology for showing behavioural equivalence on the basis of our AR. We symbolically verify equations derived from our AR, automatically calculate the precision ε of the approximate bisimulation, and provide proof obligations that address differing behaviour at control flow elements of the model.

4. *Symbolic Verification and Automated Epsilon Calculation* We automate our methodology for a subset of Simulink models whose AR contains only *linear* differential and difference equations. We calculate all possible data flows, partition the data flows, and perform partition-wise symbolic verification with the help of a Computer Algebra System. We combine the results to obtain symbolic equivalence for the overall model. Moreover, we automatically calculate a local precision ε_l of the approximate bisimulation at the point where the refactoring took place, and propagate it through the model to obtain a global precision ε_g that is valid for the overall model.

We have published our work as follows: In [SHGG15, SHGG16b], we have introduced the AR, the denotational semantics, and have shown the soundness with the operational semantics [BC12]. Moreover, we have adapted the approximate bisimulation to Simulink and laid out how to prove behavioural equivalence for purely time-discrete and purely time-continuous models together with a calculation of the precision ε of the approximate bisimulation in these cases. In [SHGG16a], we have extended our proof scheme to cope with hybrid Simulink models with control flow in the form of Switch blocks. To this end, we have added additional proof obligations for the control signal of the Switch block that exclude time-delayed behaviour. We have described our overall approach comprehensively in [SHGG18].

With our approach, we enable designers of embedded systems to verify refactorings on data flow-oriented hybrid control models on a largely automated basis, and to gain control over the precision of the disturbance that is caused by the approximate nature of such models. Our approach is a substantial contribution to enhance the safety of embedded systems. Unlike approaches for testing, which are not able to explore the whole state space, we formally prove equivalent behaviour before and after a model transformation. Approaches for formal verification are of high importance in safety-critical environments, i.e., where either life is at stake or at least loss or erroneous behaviour is unacceptable due to tremendous costs. Examples for safety-critical systems with impact on life are pilot systems, braking and steering systems, anti-collision systems, infusion pumps, pacemaker, and defibrillators. Examples for safety-critical systems for which erroneous behaviour would be unacceptable even though not life-threatening are satellite control systems, engine control in cars, and adaptive cruise control systems.

The rest of this thesis is structured as follows. In Chapter 2, we introduce various concepts that are necessary for the understanding of this thesis. In Chapter 3, we discuss related work and distinguish this thesis from this research. In Chapter 4, we provide an overview over our approach. Chapters 5 to 7 contain the details of our approach. We start with the definition of the Abstract Representation (AR),

our denotational semantics, and the proof of soundness of it with the operational semantics in Chapter 5. We continue with the adaptation of the approximate bisimulation, our equivalence notion in Chapter 6. There, we also provide sufficient conditions to conclude approximate bisimulation for Simulink models. In Chapter 7, we describe our automation approach. In Chapter 8, we provide some details on our implementation, introduce our case studies, and discuss experimental results. We close our thesis with a conclusion and discussion of future work in Chapter 9.

Chapter 2

Background

In this chapter, we provide preliminaries that are the foundation of this thesis. We start with a brief introduction of Model Refactorings, particularly for Simulink. We proceed with mathematical preliminaries, and we close with an introduction to the semantics of Simulink.

2.1 Model Refactorings in Model Driven Engineering

Model Driven Engineering (MDE) has evolved from object-oriented development. While the main principle in object-oriented development is *everything is an object*, the main principle in MDE is *everything is a model*. A model is a simplified view of reality [Sel03]. In other words, a model is a set of statements describing the behaviour of something being developed for a specific purpose. The idea is that a model abstracts from the reality due to one of the following reasons.

- Reality is too complex to be able to describe it entirely.
- It would be too expensive to describe reality entirely.
- The full set of desired features and behaviour of the targeted product is not entirely known. In this case, the development process follows a strategy of iterative refinements, i.e., models developed in the process move from high abstraction to increasing concreteness.

The central artifact in MDE is the model. It defines the desired behaviour of the targeted product. In MDE, model transformations play an important role. The main reasons for this are [GDD+09]:

- Nowadays' products are too complex to be properly represented in a single model. Therefore, various parts covering certain aspects of the product are modelled in parallel. Transformations help to patch the diverse parts together in an overall architecture.

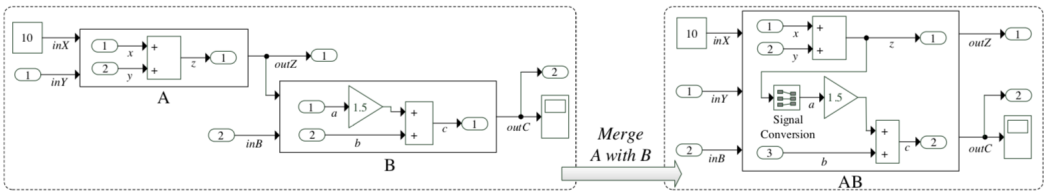

FIGURE 2.1: Example for a Simulink Refactoring: Subsystem Merge

- The systems often comprise multiple modelling languages because various domain experts are required in the development. Transformations help to map the various parts together.
- Reusability of models is very important. Consequently, transformation engines are used to automatically generate executable code from the models.
- Models become very complex and often violate various modelling guidelines. This is particularly important for safety-critical environments. For instance, ISO 26262 [Int11] is very important for MDE in the automotive sector. It defines a risk assessment for model parts, which yields a criticality for those parts. Based on the criticality, modelling guidelines are defined. During MDE, models often violate these guidelines. To control the ever-increasing complexity of models on the one hand and to establish on the other hand compliance with modelling guidelines, model refactoring approaches play an important role. Refactorings are model transformations that ensure that the behaviour before and after the transformation is equivalent.

This thesis focusses on refactorings for signal-oriented languages capable of modelling hybrid control systems. We therefore address a major business case of modern embedded systems development. As major representative we have chosen MATLAB / Simulink to demonstrate our approach of a verification methodology for behavioural equivalence of source and refactored target model on signal flow-oriented modelling languages.

An example for an often occurring Simulink refactoring is a merge of subsystems. Figure 2.1 provides this example. The figure is taken from the article [TWD13]. Submodels can be grouped into subsystems in Simulink. Such subsystems are Simulink submodels that are embedded in a surrounding Simulink model. Subsystems can in turn contain subsystems. A Simulink model can therefore contain a hierarchy of subsystems. A subsystem merge is an operation that combines the content of two subsystems into a single subsystem. The model transformation must ensure that the connections in the resulting model are properly established to guarantee equivalent

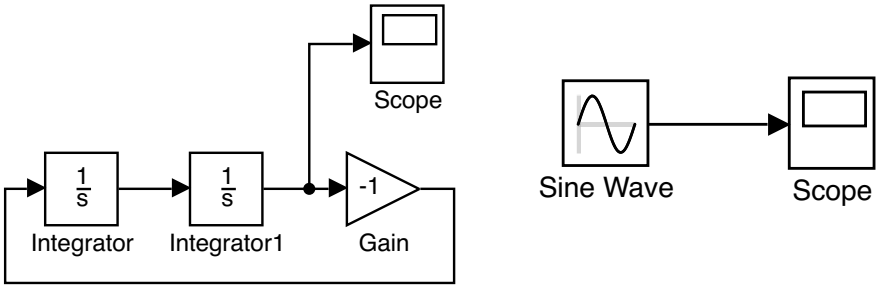

FIGURE 2.2: Example for a Simulink Refactoring: Solution of an ODE

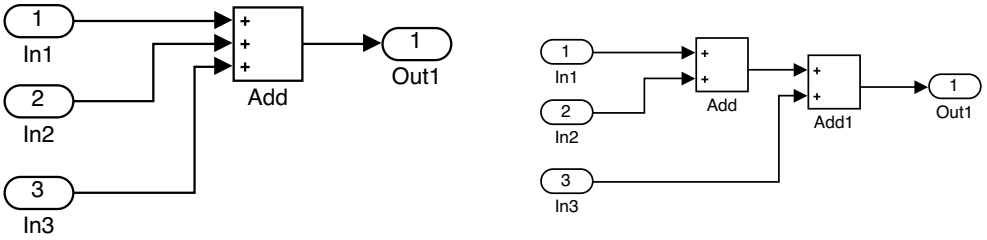

FIGURE 2.3: Example for a Simulink Refactoring: Adder Replacement

behaviour. Subsystems do not have an impact on the Simulink semantics. They are means to visually group entities of the model together. A refactoring like subsystem merge is therefore a representative of a *syntactical* refactoring.

Another example for a Simulink refactoring is given in Figure 2.2. There, the model on the left side represents an ordinary differential equation (ODE). This ODE can analytically be solved, i.e., its solution can be expressed by a closed mathematical expression, in this case by $\sin(t)$. Such a model transformation is an example of a *semantical* refactoring. Syntactically, both models are very different from each other. However, mathematically interpreted, both models yield the same values because one is the solution of the other. Actually, the execution by Simulink does not yield the exact same values because of the numerical approximation performed by Simulink. Providing conditions to show behavioural equivalence in case of syntactical and semantical refactorings and estimates for the maximal deviation of the values due to numerical approximation are major contributions of this thesis.

Figure 2.3 depicts a third example. The source model contains an Adder block with three entries, the refactoring splits this Adder into two blocks with two entries each. It is an example that establishes compliance with modelling guidelines. From our industrial partners, we know that companies often request to restrict arithmetic blocks in models to two entries.

2.2 Mathematical Foundations

In this section, we cover mathematical preliminaries. We start with some concepts from topology. They help us to understand concepts of time-discreteness and time-continuity and are used for various notions of continuity. We also introduce metrics and metric spaces in this subsection. These are the formal foundation to enable measurement of distances of observations of Simulink models. We proceed by introducing sequences and difference equations. We use these concepts to describe time-discrete behaviour of Simulink models. For the time-continuous part, we introduce topics from mathematical analysis, differentiation, and differential equations. Then, we introduce Laplace transform and z-Transform, which we use in our automation approach to obtain expressions that can be checked automatically for symbolic equivalence. We close our overview of mathematical preliminaries with an introduction in numerical mathematics, which we use to cope with the approximate nature of the semantics of Simulink.

2.2.1 Topological Preliminaries

In this thesis, notions like time-discreteness or time-continuity play an important role. This thesis is closely related to the area of mathematical analysis, differential and difference equations, and numerical mathematics. We introduce some basic topological concepts that are the foundation of various theorems we use in our approach. The material presented in the following is based on [Con10, Tim08].

We start with the most basic concept, topological spaces.

Definition 1 (Topological Space)**.** *Let X be a set, $T \subseteq \mathcal{P}(X)$. T is called a topology of X if and only if the following holds.*

- $\emptyset \in T, X \in T$
- $A_1, ..., A_n \in T \Rightarrow \bigcap_{\nu=1}^{n} A_\nu \in T$ *(closed under finite intersection)*
- *If I is an index set (countable) and $\forall \nu \in I : A_\nu \in T \Rightarrow \bigcup_{\nu \in I} A_\nu \in T$ (closed under arbitrary union)*

The elements of T are called open sets. (X, T) is called a topological space.

If the open sets are clear from the context, we can abbreviate (X, T) to X. The complements of open sets are called *closed* sets.

Definition 2 (Neighbourhood)**.** *Let (X, T) be a topological space and $p \in X$. A neighbourhood of p is a set $V \subseteq X$ that contains an open set $U \in T$ that contains p. Formally: V*

is a neighbourhood of p if

$$\exists U \in T : p \in U \wedge U \subseteq V$$

If V is an open set itself, i.e., $V \in T$, then V is called open neighbourhood of p. For sets of points $S \subseteq X$, we define a neighbourhood $V \subseteq X$ of S if V is neighbourhood for all $p \in S$.

The concept of neighbourhoods enables us to identify distinguished points in topological spaces.

Definition 3 (Distinguished Points in Topological Spaces)**.** *Let (X, T) be a topological space, $S \subseteq X$, $p \in X$. p is said to be*

- *an **inner** or **interior point** of S if there exists a neighbourhood of p that is contained in S, i.e., there is an environment $V \subseteq X$ of p with $V \subseteq S$.*
- *a **limit point** of S if every neighbourhood of p contains at least one point of S different from p itself, i.e., for all $V \subseteq X$, which are neighbourhoods of p there exists $x \in S \cap V : p \neq x$*
- *an **isolated point** of S if there exists a neighbourhood of p that does not contain any other points of S, i.e., $\exists V \subseteq X$ that is an environment of p such that $\forall x \in V : x \neq p \Rightarrow x \notin S$.*
- *a **boundary point** of S if every neighbourhood of p contains at least one point of S and one point outside of S, i.e., for every neighbourhood $V \subseteq X$ of p holds that $\exists x \in V : x \in S$ and $\exists y \in V : y \notin S$. We denote the set of boundary points of S as δS.*

A very useful characterization of closed sets, which were defined as complements of open sets, is the following.

Lemma 1 (Closed Sets and Limit Points)**.** *Let (X, T) be a topological space, $S \subseteq X$. S is closed if and only if every limit point of S is contained in S.*

Proof. See for instance [For11]. □

In this thesis, we make use of notions such as time-discreteness and time-continuity. We introduce the following two notions that are the basis for the notions of time-discreteness and time-continuity in our approach.

Definition 4 (Discrete Spaces and Sets)**.** *A topological space (X, T) is said to be **discrete** if and only if no subset of X has a limit point. A subset of a topological space $M \subseteq X$ is called **discrete** if M contains no limit point.*

Discrete spaces are related with the concept of time-discreteness.Another important concept are perfect sets, which are related with the concept of time-continuity in this thesis.

Definition 5 (Dense-in-itself and Perfect Sets)**.** *A subset $S \subseteq X$ of a topological space (X, T) is said to be **dense-in-itself** if S contains no isolated points. It is called **perfect** if it is dense-in-itself and closed in X.*

The notions discrete space and perfect set are not complementary, i.e., non-discrete spaces are not necessarily perfect and vice versa. An example for a dense-in-itself set is $\mathbb{Q}$ because in every neighbourhood of a rational number there is another rational number. Hence, it does not contain isolated points. However, $\mathbb{Q}$ is not perfect because it is not closed. This is because it does not contain all of its limit points. Limit points of $\mathbb{Q}$ are also irrational numbers because in every neighbourhood of an irrational number there is a rational number.

We introduce a very important concept, the compactness of sets. Many theorems we use in our approach assume a compact set.

Definition 6 (Open Cover, Compact Set)**.** *Let $(X, \|.\|)$ be a normed space, $M \subseteq X$.*

1. *An **open cover** $\mathcal{F}$ of M is a non-empty set of open sets in $(X, \|.\|)$ with the property*

$$M \subseteq \bigcup_{Q \in \mathcal{F}} Q := \{x \in X | \exists Q \in \mathcal{F} : x \in Q\}$$

   This means, every $x \in M$ is contained in at least one $Q \in \mathcal{F}$.

2. *M is called **compact** if every open cover $\mathcal{F}$ of M contains a finite subcover, i.e., a finite subset $\mathcal{F}_0 \subseteq \mathcal{F}$ that is an open cover of M as well.*

Metric and Normed Spaces

Topological spaces are very general concepts. In this thesis, we require some additional structure. Specifically, we need to be able to measure the distances between observations. This concept is called a *metrical function* or *metrics*, the spaces on which it operates are called *metric spaces*.

Definition 7 (Metric Space)**.** *Let X be an arbitrary set. A function $d : X \times X \to \mathbb{R}$ is called metric on X if for all $x, y, z \in X$ the following holds.*

- *(positive definiteness): $d(x, y) \geq 0$ and $d(x, y) = 0 \Leftrightarrow x = y$*
- *(symmetry): $d(x, y) = d(y, x)$*

- *(triangular inequality):* $d(x, y) \leq d(x, z) + d(z, y)$

(X, d) *is called a metric space if* d *is a metrics on* X*.*

In metric spaces, we can identify ε*-environments* or ε*-balls*. Intuitively, they are balls around a point with the radius ε.

Definition 8 (ε-environment)**.** *Let* (X, d) *be a metric space,* $x \in X, \varepsilon > 0$*. Then*

$$B_\varepsilon(x) := \{y \in X | d(x, y) < \varepsilon\}$$

is called the ε*-environment of* x *or open* ε*-ball around* x*.*

The set of ε-balls for arbitrary values of ε and all points $p \in X$ in a metric space X forms a topology, i.e., (X, T) with $T := \{B_\varepsilon(p) | p \in X, \varepsilon \in \mathbb{R}\}$ is a topological space, the ε-balls are the open sets.

A special case of metric spaces are normed spaces. Normed spaces are vector spaces over $\mathbb{R}$ or $\mathbb{C}$.

Definition 9 (Normed Space)**.** *Let* X *be a vector space over the field* $\mathbb{F} = \mathbb{R}$ *or* $\mathbb{F} = \mathbb{C}$*. A map* $\|.\| : X \to \mathbb{R}$ *is called a norm on* X *if the following holds.*

- *(positive definiteness) -* $\forall x \in X : \|x\| \geq 0$ *and* $\|x\| = 0 \Leftrightarrow x = 0$*.*
- *(homogeneity) -* $\forall \lambda \in \mathbb{F} \forall x \in X : \|\lambda x\| = |\lambda| \|x\|$*.*
- *(triangular inequality) -* $\forall x, y \in X : \|x + y\| \leq \|x\| + \|y\|$*.*

$(X, \|.\|)$ *is called a normed space if* X *is a vector space and* $\|.\|$ *is a norm on* X*.*

Normed spaces are always metric spaces because $d(x, y) := \|x - y\|$ induces a metric. We can therefore also transfer the ε-environment concept to normed spaces using $B_\varepsilon(x) := \{y \in X | \|x - y\| < \varepsilon\}$. Typical norms on $\mathbb{R}^n$ as vector space over $\mathbb{R}$ are the maximum norm $\|x\|_\infty = \max(x_1, ..., x_n)$ (x is the n-dimensional vector with components x_i), the sum norm $\|x\|_1 = \sum_{\nu=1}^{n} |x_\nu|$, or the Euclidean norm $\|x\|_2 = \sqrt{\sum_{\nu=1}^{n} x_\nu^2}$. The indices $\|.\|_\infty, \|.\|_1$, or $\|.\|_2$ are used to distinguish between different norms. If it is clear what is meant, the indices can be omitted. For vector spaces whose elements are sequences $(a_n)_n \in \mathbb{R}^{\mathbb{N}}$, the supremum norm $\|(a_n)_n\|_\infty = \sup_{n \in \mathbb{N}} |a_n|$ is a typical representative.

In this thesis, we use elements from $\mathbb{R}^n$ mainly to express the evolution of values over time for systems with n stateful blocks, e.g. Integrators or Unit Delays. In our approach, these values can be disturbed if compared between source and refactored target model. Since we are interested in the largest deviation, the maximum or supremum norm are of utmost importance in this thesis.

A special case of norms are matrix norms.

Definition 10 (Matrix Norm). *A **Matrix norm** is a norm on $\|.\| : \mathbb{R}^{n\times n} \to \mathbb{R}$, i.e., for which positive definiteness, homogeneity, and triangular inequality hold. In addition, submultiplicativity must hold, i.e.,*

$$\forall A, B \in \mathbb{R}^{n\times n} : \|A \times B\| \leq \|A\| \cdot \|B\|$$

An important example for a matrix norm that we use in this thesis is the maximum norm

$$\|A\|_\infty := \max_{1\leq i\leq n} \sum_{j=1}^{n} |a_{ij}|$$

for a matrix $A = (a_{ij})_{1\leq i,j\leq n}$. We use this maximum norm to obtain an estimate for the Jacobi matrix, which is the differential of multi-dimensional functions. Such an estimate in turn is used to calculate the diameter of the ε-tube in which the values of source and refactored target model vary.

2.2.2 Sequences and Difference Equations

In this subsection, we briefly introduce sequences and difference equations. We use difference equations to describe the behaviour of time-discrete models. Sequences are solutions of difference equations.

Definition 11 (Sequences). *Let X be some space (topological, metric, or normed). A sequence is a mapping*

$$\mathbb{N} \to X, i \mapsto a_i$$

We therefore refer to a single element from the sequence with a_i and to the whole sequence as $(a_n)_n$.

An important property of a sequence is the convergence to a limit.

Definition 12 (Limit of a Sequence). *Let $(a_n)_n$ be a sequence in a metric space (X, d). $x \in X$ is said to be a limit point of the sequence if and only if*

$$\forall \varepsilon > 0 \exists n_0 \forall n \geq n_0 : d(a_n, x) < \varepsilon$$

We denote this with

$$\lim_{n\to\infty} a_n = x$$

The concepts of sequences and limits allow us to obtain an equivalent statement for compactness: Compact sets always contain limits of sequences that consist of members of the set. Limits of sequences containing elements of non-compact sets

may be outside of the set. The next lemma therefore provides a very intuitive view for compactness.

Lemma 2 (Sequential Compactness). *Let $(X, \|.\|)$ be a normed space and $\emptyset \neq M \subseteq X$. M is compact if and only if M is **sequentially compact**, i.e., every sequence in M contains a subsequence with a limit in M.*

Proof. See for instance [For11] □

In $\mathbb{R}^n$, compactness has another very intuitive interpretation described in the theorem of Heine-Borel.

Lemma 3 (Heine-Borel Theorem). *Let $\emptyset \neq M \subseteq \mathbb{R}^n$. M is compact if and only if M is closed and bounded. A set $\emptyset \neq M \subseteq \mathbb{R}^n$ is bounded if there exists an $r > 0$ such that $M \subseteq B_r(0)$, i.e., M can be enwrapped by a sphere with finite radius.*

Proof. See for instance [For11] □

In this thesis, time-discrete systems are described by difference equations, which we introduce in the following. The theory can be found e.g. in [KN12].

Definition 13 (Difference Equation). *Let $D \subseteq \mathbb{R}^n$ and $f : \mathbb{N} \times D \to \mathbb{R}$ be a function. A **difference equation** of first order is an equation of the form*

$$y_{n+1} = f(n, y_n)$$

where y_0 is given (the initial value). A solution is a sequence $(a_n)_n$ fulfilling the equation, i.e.,

$$\forall n \in \mathbb{N}, n \geq 1 : a_{n+1} = f(n, a_n) \wedge a_0 = y_0$$

According to [Tau64], difference equations have always a unique solution. The reason is that the value $a_1 = f(1, a_0)$ is uniquely determined since f is a function and per assumption defined on $\mathbb{N} \times D$. The same applies to $a_2, a_3, ...$ The proof is performed by natural induction.

In this thesis, we pay special attention to linear difference equations, which we introduce in the following.

Definition 14 (Linear Difference Equation Systems). *Let $A : \mathbb{N} \to \mathbb{R}^{n \times n}$ a continuous function ($\mathbb{R}^{n \times n}$ denotes the space of real-valued $n \times n$ matrices), $b : \mathbb{N} \to \mathbb{R}^n$ a continuous function. Then*

$$y_{n+1} = Ay_n + b, y_0 \text{ given}$$

*is called a **linear difference equation system**. Note that A can also only contain real values, i.e., constants.*

Linear difference equation systems are also always uniquely determined. In our automation approach, we verify behavioural equivalence for subsets of Simulink models whose behaviour can be expressed by linear difference or differential equations.

2.2.3 Functions, Functional Analysis, and Differential Equations

For the time-continuous part of models, we deal with continuous and also with differentiable functions. In this subsection, we recapture the basic notions for such functions. We denote the set of functions $A \to B$ with B^A.

We start with various forms of continuity. Continuity of functions is a necessary prerequisite in many theorems we use in this thesis.

Definition 15 (Continuous Function)**.** *Let $M \subseteq \mathbb{R}$, $f : M \to \mathbb{R}$ be a function, $a \in M$. f is said to be continuous at a if for every neighborhood V of $b := f(a)$ there exists a neighborhood U of a such that*

$$f(U \cap M) \subseteq V, \text{ i.e., } \forall x \in U \cap M : f(x) \in V$$

f is said to be continuous on M if it is continuous for all $a \in M$. We denote the set of continuous functions $M \to \mathbb{R}$ with $C(M, \mathbb{R})$.

The following lemma provides two equivalent statements for continuity. The first one intuitively translates to: A function is continuous if and only if the arguments are close enough, the values of the function must be close to each other as well. The second statement connects limits of sequences of values of a function with continuity.

Lemma 4. *Alternative Continuity Definitions for Functions Let $f : M \to \mathbb{R}$ be a function, $a \in M$. Equivalent forms for continuity of f at a are*

1. $$\forall \varepsilon > 0 \exists \delta > 0 \forall b \in M : |a - b| < \delta \Rightarrow |f(a) - f(b)| < \varepsilon$$

2. *For every sequence $(a_n)_n$ in M with $\lim_{n\to\infty} a_n = a$ the following holds*

$$\lim_{n\to\infty} f(a_n) = f(a) = f\left(\lim_{n\to\infty} a_n\right)$$

Proof. See for instance [For11] □

In the following definition, we introduce a stronger form of continuity, the uniform continuity. While continuity is explained for a single point a in the domain

of a function, uniform continuity applies either to the whole domain or not. The main difference to continuity is that the ε, which can be determined for every single point to establish continuity, must hold for all points in the function's domain for uniform continuity. Uniform continuity is the type of continuity resulting in an important theorem, the Theorem of Arzelà-Ascoli. We use this to prove soundness of our Abstract Representation, which is an equation-based representation of Simulink models, with the operational semantics.

Definition 16 (Uniform Continuity of Functions)**.** *Let (X, d_X) and (Y, d_Y) be metric spaces. A function $f : X \to Y$ is said to be **uniformly continuous** if*

$$\forall \varepsilon > 0 \exists \delta > 0 \forall a, b \in X : d_X(a, b) < \delta \Rightarrow d_Y(f(a), f(b)) < \varepsilon$$

Lipschitz continuity, which we introduce in the next definition, is another form of continuity. It is a sufficient precondition for uniqueness of solutions of ordinary differential equations, which we use to describe the behaviour of time-continuous models. Intuitively, a function is Lipschitz continuous if its growth is bounded.

Definition 17 (Lipschitz Continuity)**.** *Let (X, d_X) and (Y, d_Y) be metric spaces. A function $f : X \to Y$ is said to be **Lipschitz-continuous** if a constant $L \in \mathbb{R}, L \geq 0$ exists with*

$$\forall a, b \in X : d_Y(f(a), f(b)) \leq L \cdot d_X(a, b)$$

A weaker condition is local Lipschitz continuity. A function is locally Lipschitz continuous if the Lipschitz condition holds in an environment for every point of the function's domain.

Definition 18 (Local Lipschitz Continuity)**.** *Let (X, d_X) and (Y, d_Y) be metric spaces. A function $f : X \to Y$ is said to be **locally Lipschitz-continuous** if for every $x \in X$ there exists an environment U of x such that there exists a constant $L \in \mathbb{R}, L \geq 0$ with*

$$\forall a, b \in U : d_Y(f(a), f(b)) \leq L \cdot d_X(a, b)$$

In other words, the Lipschitz condition must hold in the environment.

The next notion we introduce is the equicontinuity of functions. Unlike other continuity notions, equicontinuity is a concept that refers to sequences of functions rather than points in function's domains. We use the equicontinuity as a precondition to apply the Theorem of Arzelà-Ascoli, which we use to prove soundness of our Abstract Representation with the operational semantics.

Definition 19 (Equicontinuity of Functions)**.** *Let (X, d_X) and (Y, d_Y) be metric spaces, $\mathcal{F} \subseteq Y^X$ a family (i.e., a set) of functions. $\mathcal{F}$ is said to be **equicontinuous** if*

$$\forall \varepsilon > 0 \exists \delta > 0 \forall f \in \mathcal{F} \forall a, b \in X : d_X(a, b) \leq \delta \Rightarrow d_Y(f(a), f(b)) \leq \varepsilon$$

Equicontinuity extends the concept of uniform continuity to a family of functions. If $\mathcal{F}$ consists only of one function, equicontinuity and uniform continuity are the same. If there is more than function in $\mathcal{F}$, equicontinuity requires uniform continuity for all functions in $\mathcal{F}$ and moreover that δ can be chosen depending on ε only, i.e., without dependence on the functions in $\mathcal{F}$.

After having introduced the various forms of continuity, we define two forms of convergence of sequences of functions. We start with the pointwise convergence, which is intuitively the convergence of the sequence of values at a point in the function's domain. It yields a limit function.

Definition 20 (Pointwise Convergence of Function Sequences)**.** *Let (X, d_X) and (Y, d_Y) be metric spaces, $(f_n)_n : X \to Y$ be a sequence of functions. The sequence $(f_n)_n$ is said to be **pointwise convergent** to a function $f : X \to Y$ if the sequence of values $f_n(t)$ converges to $f(t)$ for all $t \in X$. In symbols*

$$\forall t \in X : \lim_{n \to \infty} f_n(t) = f(t)$$

The uniform convergence, which we introduce next is a stronger form of convergence than pointwise convergence. Intuitively, an ε must suffice for all functions and all values of the function's domain to keep them in a distance of at most ε around the limit function. A big advantage of uniform convergence over pointwise convergence is that continuity of the functions in the sequence transfers to the limit function. We exploit this when we show the soundness of our Abstract Representation with the operational semantics for time-continuous models.

Definition 21 (Uniform Convergence of Function Sequences)**.** *Let (X, d_X) and (Y, d_Y) be metric spaces, $(f_n)_n : X \to Y$ be a sequence of functions. The sequence $(f_n)_n$ is said to be **uniformly convergent** to a function $f : X \to Y$ if*

$$\forall \varepsilon > 0 \exists n_0 \in \mathbb{N} \forall x \in X \forall n \geq n_0 : d_Y(f_n(x), f(x)) < \varepsilon$$

Equivalent to that, we can also formulate that

$$\lim_{n \to \infty} \sup_{x \in X} d_Y(f_n(x), f(x)) = \lim_{n \to \infty} \|f_n - f\|_\infty = 0$$

The latter equation only applies if Y is also a normed space with supremum metric $\|.\|\infty$.

Some of the theorems we use in this thesis require a function to be bounded, i.e., the values of a function must be finite for all points in the function's domain.

Definition 22 (Boundedness of Functions)**.** *Let (X, d_X) and (Y, d_Y) be metric spaces, $\mathcal{F} \subseteq Y^X$ a family (i.e., a set) of functions. $\mathcal{F}$ is said to be*

- ***point-wise bounded** if*

$$\forall x \in X \exists M \in \mathbb{R} \forall f \in \mathcal{F} : \|f(x)\| \leq M$$

- ***uniformly bounded** if*

$$\exists M \in \mathbb{R} \forall x \in X \forall f \in \mathcal{F} : \|f(x)\| \leq M$$

The next theorem, the Theorem of Arzelà-Ascoli [Tim08], provides conditions for which a sequence of functions on a compact domain converge uniformly. We use this theorem in our soundness proof of our Abstract Representation with the operational semantics.

Theorem 1 (Theorem of Arzelà-Ascoli)**.** *Let $\mathcal{F} \subseteq C(M, \mathbb{R})$ be a family of continuous functions, $M \subseteq \mathbb{R}$ a compact set. Then every sequence of $\mathcal{F}$ contains a uniformly convergent subsequence if and only if $\mathcal{F}$ is point-wise bounded and equicontinuous. In this case, $\mathcal{F}$ is even uniformly bounded.*

Differentiation and Differential Equations

In this subsection, we introduce differentiation for multidimensional functions. In this thesis, we use this concept in various situations. It is the foundation for differential equations, which we use to describe behaviour of time-continuous models. Moreover, we use the multidimensional differential, the Jacobi matrix, in the calculation of the maximal difference ε of values from source and refactored target model.

Definition 23 (Differentiable Functions, Differential, Jacobi Matrix)**.** *Let $a \in M \subseteq \mathbb{R}^n$ and $f \in (\mathbb{R}^m)^M$.*

1. *f is called **differentiable in** a if M is a neighbourhood of a and there exists a matrix A such that*
$$\lim_{x \to a} \frac{f(x) - (f(a) + A(x - a))}{\|x - a\|} = 0$$
2. *Is $\emptyset \neq E \subseteq M$ then f is called **differentiable on** E if f is differentiable in every point of E.*

3. *f is called **differentiable** if f is differentiable on M.*

*The matrix A is uniquely determined. It is called the **differential** of f. If f is differentiable in a, then the differential is determined by the **Jacobi Matrix**.*

$$\mathcal{J}_f(a) = \left(\frac{\delta f_i}{\delta_{a_j}}(a)\right)_{1\leq i\leq m, 1\leq j\leq n} = \begin{pmatrix} \frac{\delta f_1}{\delta a_1}(a) \dots \frac{\delta f_1}{\delta a_n}(a) \\ \vdots \quad \vdots \\ \frac{\delta f_m}{\delta a_1}(a) \dots \frac{\delta f_m}{\delta a_n}(a) \end{pmatrix}$$

We also may write $f'(a) = \mathcal{J}_f(a)$ or $\frac{d}{dx}f(a)$ to denote the derivative.

Note that in this formula the following implicitly was assumed.

$$f(a) = \begin{pmatrix} f_1(a) \\ \vdots \\ f_n(a) \end{pmatrix}, a = \begin{pmatrix} a_1 \\ \vdots \\ a_n \end{pmatrix}$$

The formulas $\frac{\delta f_i}{\delta a_j}(a)$ are the **partial derivatives**.

We briefly capture some rules for differentiation, which we use for calculation of the ε, the maximal distance of values of source and refactored target model.

Lemma 5 (Rules for Differentiation)**.** *Let $a \in M \subseteq \mathbb{R}^n$.*

1. ***Linearity:** If $f, g \in \mathbb{R}^{m^M}$ are differentiable in a and $\alpha \in \mathbb{R}$, $f + g$ and αf are differentiable in a as well with*

$$(f + g)'(a) = f'(a) + g'(a), (\alpha f)'(a) = \alpha f'(a)$$

2. ***Product Rule:** If $f, g \in \mathbb{R}^{m^M}$ are differentiable in a, then fg is differentiable in a as well with*

$$(fg)'(a) = g(a)f'(a) + f(a)g'(a)$$

3. ***Chain Rule:** If $f \in \mathbb{R}^{m^M}$ is differentiable in a, $N \subseteq \mathbb{R}^m$ with $f(M) \subseteq N$, and $g \in \mathbb{R}^{l^N}$ is differentiable in $b := f(a)$, then $g \circ f$ is differentiable in a with*

$$(g \circ f)'(a) = g'(f(a))f'(a)$$

Proof. See e.g. [For11] □

The following lemma, the Mean Value Theorem, intuitively means that there is a point in an open set where a function is differentiable whose differential is parallel to

the secant connecting the function's values at the edge of the open set. We provide it for the multidimensional case. The Mean Value Theorem is used as precondition in various statements.

Lemma 6 (Mean Value Theorem). *Let $\emptyset \neq M \subseteq \mathbb{R}^n$ be an open set, $a, b \in M$ such that*

$$]a, b[:= \{a + t(b - a) | 0 < t < 1\} \subseteq M$$

which is the connecting line between a and b, is contained in M. Is $f : M \to \mathbb{R}^n$ is continuous on $[a, b] = \{a + t(b - a) | 0 \leq t \leq 1\}$ and differentiable on $]a, b[$, then there exists $z \in]a, b[$ such that

$$f(b) - f(a) = f'(z)(b - a)$$

Proof. See e.g. [For11] □

In the following, we define ordinary differential equations. We use this concept to describe the behaviour of time-continuous Simulink models.

Definition 24 (First Order Ordinary Differential Equation). *Let $M \subseteq \mathbb{R}^{n+1}$, $f : M \to \mathbb{R}^n$, $(t_0, \boldsymbol{y}_0) \in M$. An **ordinary differential equation of first order** (ODE) is of the form*

$$\frac{d}{dt}\boldsymbol{y}(t) = f(t, \boldsymbol{y}(t)), \mathbf{y}(t_0) = \mathbf{y}_0$$

*In this definition, we denote vectors with bold variables, i.e., $\mathbf{y}(t) = \begin{pmatrix} y_1(t) \\ \vdots \\ y_n(t) \end{pmatrix}$. A **solution** of the ODE is a function $\boldsymbol{\lambda} : \mathcal{I} \to \mathbb{R}^n$ on an interval $\mathcal{I} \subseteq \mathbb{R}$ that fulfills the ODE equations, i.e.,*

$$t_0 \in \mathcal{I} \wedge \forall t \in \mathcal{I} : \frac{d}{dt}\boldsymbol{\lambda}(t) = f(t, \boldsymbol{\lambda}(t))$$

Figure 2.4 shows a vector plot of an ODE and a solution. There, at each point (t, y), a vector arrow in direction $(1, f(t, y))$ is depicted. The vector arrows indicate the ascending slope for the solution at that particular point. A solution runs through the initial point (t_0, y_0) and fits in the vector plot as shown in Figure 2.4. An important question is under which circumstances a solution exists for an ODE and when it is unique. In this thesis, we describe time-continuous behaviour via ODEs. In order to show behavioural conformance of source and refactored target model, we reason on solutions of ODEs. We provide sufficient conditions in our methodology to conclude approximate behavioural equivalence. One of these conditions is the uniqueness of the solution of ODEs. In the following two Theorems, we provide conditions for existence and uniqueness of solutions for ODEs.

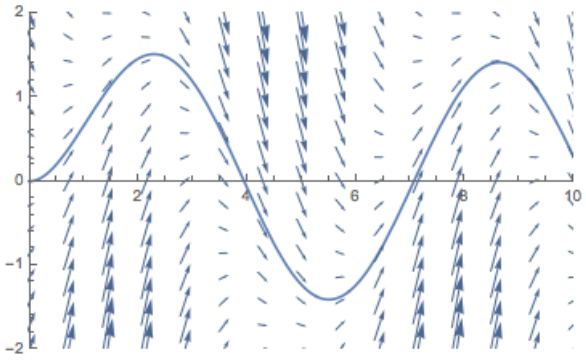

FIGURE 2.4: Vector Plot for an ODE

Theorem 2 (Existence Theorem of Peano). *Let $f : [a, b] \times \overline{B}_R(y_0) \to \mathbb{R}^n$ be a continuous function with $\overline{B}_R(y_0) = \{y \in \mathbb{R}^n | \|y - y_0\| \leq R\}$. Let furthermore*

$$\frac{d}{dt}y(t) = f(t, y), y(a) = y_0$$

be an ordinary differential equation and

$$M := \max\{\|f(t, y)\| | (t, y) \in [a, b] \times \overline{B}_R(y_0)\}$$

*This means, M is the maximal value of f on the compact set (which always exists due to the Extreme Value Theorem (cf. [For07]) for continuous functions). Then, this ODE has a **solution** on $[a, a + \alpha]$ where*

$$\alpha := \min\left(b - a, \frac{R}{M}\right)$$

The case $M = 0$ is included by setting $\frac{R}{0} := \infty$.

Proof. See e.g. [Aul04]. □

This includes the cases where f is defined on $[a, b] \times \mathbb{R}^n$ and continuous. Then, the solution exists on the whole interval $[a, b]$.

Theorem 3 (Existence and Uniqueness Theorem of Picard-Lindelöf). *Let $f : S \to \mathbb{R}^n$ with $S := [a, b] \times \overline{B}_R(y_0)$ be a locally Lipschitz-continuous function in y, i.e.,*

$$\exists L \geq 0 \forall t \in (t, y_1), (t, y_2) \in S : \|f(t, y_1) - f(t, y_2)\| \leq L\|y_1 - y_2\|$$

Let furthermore

$$\frac{d}{dt}y(t) = f(t, y), y(a) = y_0$$

be an ordinary differential equation and

$$M := \max\{\|f(t,y)\| | (t,y) \in [a,b] \times \overline{B}_R(y_0)\}.$$

Then, this ODE has a ***unique solution*** *on* $[a, a+\alpha]$ *where*

$$\alpha := \min\left(b-a, \frac{R}{M}\right).$$

If $S = [a,b] \times \mathbb{R}^n$ *and the Lipschitz condition holds globally there, then there exists a unique solution on the whole interval* $[a,b]$.

Proof. See e.g. [Aul04]. □

The following lemma provides means to calculate the Lipschitz constant, which is the bound for the maximal increase of a function. This constant in turn is used for the calculation of the ε, the maximal distance of values of source and refactored target model.

Lemma 7 (Sufficient Condition for Lipschitz Continuity)**.** *Let* $U \subseteq \mathcal{I} \times O$, $\mathcal{I} \subseteq R$ *an interval,* $O \subseteq \mathbb{R}^n$ *an open set, and* $f : U \to \mathbb{R}^n$ *a function for which all partial derivatives* $\frac{\delta f_i}{\delta y_j}$ *are continous, i.e.,* f *is said to be continuously partially differentiable with respect to* $y = (y_1, ..., y_n)^T$. *Then* f *is locally Lipschitz continuous. If* O *is convex and* $\|\mathcal{J}_f(y)\|_\infty$ *is bounded on* U, *then* f *is globally Lipschitz continuous with respect to* y. *The value*

$$L := \|\mathcal{J}_f(y)\|_\infty$$

is a suitable Lipschitz constant.

Proof. See e.g. [For11, Aul04]. The proof for the estimate of L uses the Mean Value Theorem. □

In our automation approach, we pay special attention on linear ODEs.

Definition 25 (Linear ODEs)**.** *Let* $\mathcal{I} \subseteq \mathbb{R}$ *be an interval,* $A : \mathcal{I} \to \mathbb{R}^{n \times n}$ *a continuous function (*$\mathbb{R}^{n \times n}$ *denotes the space of real-valued* $n \times n$ *matrices),* $b : \mathcal{I} \to \mathbb{R}^n$ *a continuous function. Then*

$$\frac{d}{dt}y = A(t)y + b(t), y(t_0) = y_0$$

is called a ***linear ODE****. Note that* A *can also only contain real values, i.e., constants.*

The solutions of linear ODEs always exist and are unique. This follows directly from Theorem 3 and Lemma 7.

2.2.4 Laplace Transform and z-Transform

In this subsection, we briefly introduce Laplace transform and z-transform. We use these in our automation approach. They allow to represent ODEs and difference equations as algebraic expressions. Such algebraic expressions can be automatically checked for equivalence via symbolic unification with the help of a Computer Algebra System.

Laplace Transform

A Laplace transform is a function that operates on a vector space of functions mapping them to functions. The original function f is in this context sometimes referred to as the function in time space, the Laplace transformed $\mathcal{L}(f)$ as the respective function in the frequency domain. This terminology originates from the Fourier series theory. There, periodical functions can be represented as infinite sum of weighted sine and cosine waves. The Fourier theory is extended to cover non-periodical signals as well. The coefficients for the frequencies of the involved sine and cosine waves are calculated by an expression that is very similar to the Laplace transform. The Laplace transform is a more generalised version of that. Formally, the Laplace transform can be defined as follows [Gra12, Kai80].

Definition 26 (Laplace Transform)**.** *Let $f : [0, \infty[\rightarrow \mathbb{R}$ a function with $\forall t < 0 : f(t) = 0$. The Laplace transform of f is defined as*

$$\mathcal{L} : C^0([0, \infty[\rightarrow \mathbb{C}), \mathcal{L}(f)(s) := \int_0^\infty f(t)e^{-st}dt$$

for all $s \in \mathbb{C}$ for which this limit exists.

Since in our automation approach, our equivalence check relies on Laplace transform, we require a criterion for the existence of the Laplace transform. This uses a notion that describes a property for functions, the *at most exponential growth*.

Definition 27 (Functions of at most exponential growth)**.** *A function $f : [0, \infty[\rightarrow \mathbb{R}$ is of at most exponential growth if*
$\exists \alpha > 0 : \exists M > 0 : \exists t_0 > 0 : \forall t \geq t_0 : ||f(t)|| \leq Me^{\alpha t}$.

Lemma 8 (Existence of the Laplace Transform)**.** *Let $f : [0, \infty[\rightarrow \mathbb{R}$ be a continuous function, and of at most exponential growth. Then $\beta \geq 0$ exists such that $\mathcal{L}(f)(s)$ exists for all $s \in \mathbb{C}$ with $Re(s) > \beta$, i.e., on a complex half plane, and converges absolutely.*

The Laplace transform has some important properties that we exploit in our approach for automated equivalence checking of Simulink models.

In particular, for linear ordinary differential equations (ODEs), the Laplace transform yields algebraic equations, which can automatically be solved using Computer Algebra Systems (CAS). The formal foundation for this are *linearity* and the *first derivative property* of the Laplace transform.

Lemma 9 (Properties of Laplace Transform)**.** *Let $f, g : [0, \infty[\to \mathbb{C}$ and $\mathcal{L}(f)$ exist for s with $Re(s) > \alpha$, $\mathcal{L}(g)$ exist for s with $Re(s) > \beta$.*

- *$\mathcal{L}(f + g)$ exists for s with $Re(s) > \max(\alpha, \beta)$ and $\mathcal{L}(af + bg) = a\mathcal{L}(f) + b\mathcal{L}(g)$ (linearity).*
- *Let f be differentiable, $\frac{d}{dt}f$ continuous on $]0, \infty[$ and of exponential growth with rate at most $\alpha > 0$. Then $\mathcal{L}\left(\frac{d}{dt}f\right)(s) = s\mathcal{L}(f)(s) - f(0^+)$ for s with $Re(s) > \alpha$. Note that $f(0^+) := \lim_{t \to 0, t>0} f(t)$ (first derivative).*

In our automation approach, we transform complex ODEs via Laplace transform to algebraic expressions. We show in our approach that two expressions that are the result of the Laplace transform are equivalent, which can be done automatically with a CAS. In order to conclude that the original functions are equivalent as well, we require the injectivity of the Laplace transform.

Lemma 10 (Injectivity of Laplace Transform)**.** *Let $f, g : [0, \infty[\to \mathbb{C}$ be continuous and of exponential growth with rate at most $\alpha > 0$, $\beta \geq 0$ the left bound of the half plane of convergence. If $\forall s \in \mathbb{C} : Re(s) > \beta \Rightarrow \mathcal{L}(f)(s) = \mathcal{L}(g)(s)$, then $\forall t > 0 : f(t) = g(t)$.*

The requirement for continuity of the functions can be amended: Laplace transform is also injective for piecewise-smooth functions as described in [AG08]. A function is smooth if it is arbitrarily often continuously differentiable. It is piecewise-smooth on the interval $\mathcal{I}$ if it is smooth there except for finitely many places $t_0 < t_1, ..., < t_m$. We require this relaxation if time-discrete partitions feed into time-continuous partitions. In such cases, the signal or output function of the time-discrete partition is not continuous. However, we require the injectivity in the subsequent time-continuous partition. In essence, if we amend the prerequisite for these cases, we are able to cope with them in our approach as well.

z-Transform

The analogon of the Laplace transformation for time-discrete functions is the z-transform [EA06]. This enables to transform difference equations to algebraic expressions, which in turn enables automated symbolic equivalence checking via a Computer Algebra System in our automation approach.

Definition 28 (z-Transform). *Let $(a_k)_k$ be a sequence of complex (or real) values. The z-transformation of $(a_k)_k$ is a function $\mathscr{Z}$ that associates a power series with the sequence. For $z \in \mathbb{C}$, this power series is defined as $\mathscr{Z}((a_k)_k)(z) := \sum_{\nu=0}^{\infty} a_\nu \frac{1}{z^\nu}$.*

As for the Laplace transform, we define the notion of exponential growth for sequences, which is used in the lemma for the existence of the z-Transform.

Definition 29 (Sequences of at most exponential growth). *A sequence $(a_k)_k$ is of at most exponential growth if $\exists \alpha > 0 \exists M > 0 \forall \geq 0 : ||a_k|| \leq M\alpha^k = Me^{\ln(\alpha)k}$.*

As for the Laplace transform, we introduce conditions for the existence of the z-Transform.

Lemma 11 (Existence of the z-Transform). *If $(a_k)_k$ as above fulfills the at most exponential growth condition of rate $\alpha > 0$, then exists $\mathscr{Z}((a_k)_k)(z)$ for $z \in \mathbb{C}, ||z|| > \alpha$.*

Analogously to the Laplace transform, in our automation approach, we conclude the equivalence of the solutions of the difference equations describing the behaviour of time-discrete models, from the equivalence of z-Transformed expressions. To enable this conclusion, we require conditions for the injectivity of the z-Transform.

Lemma 12 (Injectivity of the z-Transform). *If $(a_k)_k, (b_k)_k$ as above both fulfill the at most exponential growth condition of rates $\alpha, \beta > 0$ and $\forall z \in \mathbb{C}, ||z|| > \max(\alpha, \beta) : \mathscr{Z}((a_k)_k)(z) = \mathscr{Z}((b_k)_k)(z)$. Then $\forall k \geq 0 : a_k = b_k$.*

As for Laplace transform, we introduce properties of the z-Transform. They justify the transformation of linear difference equations to algebraic expressions, which we exploit in our automation approach.

Lemma 13 (Properties of z-Transform). *Let $(a_k)_k, (b_k)_k$ sequences as above, of exponential growth of rates at most $\alpha, \beta > 0$, $c, d \in \mathbb{C}$.*

- *$\mathscr{Z}(c(a_k)_k + d(b_k)_k)$ exists for z with $||z|| > \max(\alpha, \beta)$ and $\mathscr{Z}(c(a_k)_k + d(b_k)_k) = c\mathscr{Z}((a_k)_k + d\mathscr{Z}((b_k)_k)$ (linearity)*
- *$\mathscr{Z}(0, ..., 0, a_1, ...)(z) = z^{-n}\mathscr{Z}((a_k)_k)(z)$ (a shift/delay by n positions).*

2.2.5 Numerical Mathematics Preliminaries

One of our criteria of this thesis is that we consider the approximate nature of the semantics of signalflow-oriented languages. The reason for approximation coming into focus is that although dataflow-oriented languages define blocks for time-continuous signals, these cannot actually be evaluated in a mathematically precise

manner. Tools interpreting dataflow-oriented languages always run on machines with finite space and computation duration. Therefore, they depend on numerical approximation to solve time-continuous problems, e.g. ODEs.

It should be noted that basically there are two kinds of numerical approximations: 1) deviations originating from the application of approximation techniques in order to solve time-continuous problems, e.g. use of Euler method to obtain a solution of an ODE, and 2) deviations caused by imperfect arithmetics. The latter comes into play because we deal with machine precisions, i.e., finitely many storage results in finitely many digits that can be represented by machines. However, real numbers may need infinitely many digits to represent them precisely. Hence, there is a gap, which may be intensifyed by arithmetics running on machines.

In this thesis, we only consider the former approximation errors, i.e., those resulting from approximation of time-continuous problems. This is consistent with the operational semantics that is our formal foundation. However, our approach can be extended to cope with imperfect arithmetics as well, which is left for future work.

In this section, we introduce the basic numerical concepts to provide an orientation in the domain of numerical approximation.

Basic concepts

Consider $f : X \to Y$ to be a function operating on some normed spaces X, Y. Let $x \in X$ be an undisturbed value and $\tilde{x} = x + \varepsilon$ be a disturbed value. Let furthermore be $\tilde{f} : X \to Y$ a function representing the results from the application of a numerical algorithm. The following properties are interesting to describe the relation between f and $\tilde{f}$.

- **Condition:** $\|f(\tilde{x}) - f(x)\|$ What is the impact on the original problem on disturbances?
- **Stability:** $\|\tilde{f}(\tilde{x}) - \tilde{f}(x)\|$
  How sensitive is the numerical algorithm to disturbances?
- **Consistency:** $\|\tilde{f}(x) - f(x)\|$
  How well does the numerical algorithm actually solve the problem?
- **Convergence:** $\|\tilde{f}(\tilde{x}) - f(x)\|$
  How well does the numerical algorithm solve the problem with disturbed input?

The errors in the respective category are named like the category, e.g. we speak of the consistency error, the convergence error. The first item concerns with disturbances of the original problem, the others concern with disturbances that occur due to the approximation.

Consistency plays an important role for the soundness of our Abstract Representation (AR) in Chapter 5. The AR is our mathematically precise representation of dataflow-oriented models in terms of equations, particularly ODEs for time-continuous models. We show in Chapter 5 that for time-continuous models, the sequence of executions of the operational semantics of Simulink for sample step sizes converging to 0 converges uniformly to a function that fulfills the equations of the AR. The proof makes use of the consisteny of the numerical algorithms in Simulink. We estimate a conditional error in our automation approach. There, we partition given models, and propagate local disturbances ε through subsequent partitions.

Consistency and stability play an important role in Chapter 6. There, we use them to calculate the disturbance due to numerical execution in various model types. We calculate a conditional error in 7. In this chapter, we automate our approach. There, we provide a mechanism for propagation of the local error ε_l to obtain a global error ε_g.

Single Step Approximation Techniques

Although theoretically a solution for an ODE exists in many cases, it is often not possible to obtain an analytical solution, i.e., a function that can be formulated as a closed mathemtatical expression. Therefore, solutions are approximated. Such approximations or simulations run on a computer with finite amount of space and run time. Hence, the theoretically uncountably many time steps can only be approximated in finite runtime steps.

In this section, we provide a very basic introduction to approximation techniques for ODEs. We restrict ourselves on single step techniques, particularly on the Euler method, to make the concepts transparent. There are more sophisticated techniques available, also in Simulink. However, our framework focusses on the concepts. Our framework can easily be transfered to more sophisticated techniques. For more details and a more detailed overview, e.g. the textbooks [Her04, But05, SWP12, But16] can be used for reference.

A very basic but intuitive technique to calculate an approximated solution for an ODE

$$\frac{d}{dt}y(t) = f(t, y(t)), y(t_0) = y_0$$

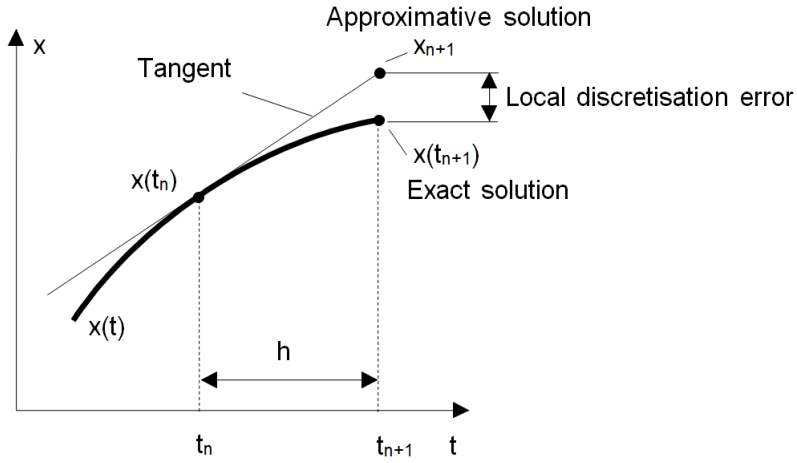

FIGURE 2.5: Euler Method Principle

is the *Euler method*. Note that y can be a multi-dimensional vector. The concepts are the same for this case. The Euler method defines a traverse, the Euler polygonal line, to construct the approximated solution. Let h be a sample step size, then we define the following sequence.

$$y_{n+1} = y_n + h \cdot f(t_n, y_n)$$

In the formula, y_0 is given by the initial condition. The time steps t_n are defined as $t_n := t_0 + n \cdot h$. Between time steps, the graph of the Euler polygonial line connects the points (t_n, y_n) and (t_{n+1}, y_{n+1}) linearly. The idea of the Euler method is that $f(t_n, y_n)$ defines the derivative of the wanted solution at the point (t_n, y_n). This is exactly the direction of the vectors $(1, f(t_n, y_n))$ in Figure 2.4. Hence, at point (t_n, y_n) we take this derivative and move along the line pointing in the direction $(1, f(t_n, y_n))$ for a distance of h until we reach the next sampling point (t_{n+1}, y_{n+1}) where the calculation starts again.

Figure 2.5 illustrates one step of the Euler method. There, x is the exact solution where $(x_n)_n$ denotes the approximation. In the figure, the consistency error is called local discretisation error, a synonymous notion. The method performs the same iteration step at x_{n+1} as at x_n. However, due to the approximation, we obtained a consistency error, which needs to be considered in the next step. This implies that in addition to the consistency error, a stability error needs to be considered. In other words, the local discretisation errors sum up to a global error. We provide more details in the next subsection.

A more general view on single step methods is that they define a process function ϕ such that

$$y_{n+1} = y_n + h_n \cdot \Phi(h, f)(t_n, y_n)$$

The Euler method is then a special case with

$$\Phi(h, f)(t, y) = f(t, y).$$

Another example is the *Heun* method with

$$\Phi(h, f)(t, y) = \frac{1}{2}(f(t, y) + f(t + h, y + hf(t, y)))$$

Very important in Simulink are also the *Runge-Kutta techniques* with the approximation being defined as follows.

$$\Phi(h, f)(t, y) = \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4), k_1 = f(t, y), k_2 = f\left(t + \frac{1}{2}h, f\left(y + \frac{1}{2}hk_1\right)\right)$$

$$k_3 = f\left(t + \frac{1}{2}h, y + \frac{1}{2}hk_2\right), k_4 = f(t + h, y + hk_3)$$

All these techniques only use one time step to calculate the next time step. That is why they are being called *single step techniques*. Multi step techniques use more sophisticated calculations to provide better approximations. However, the concepts stay the same.

Error Analysis for One Step Techniques

We need to consider the following errors that occur during the approximation.

1. The *consistency* error. This is the error that occurs during one sample step. This is connected with the *consistency* of the numerical technique as introduced in Section 2.2.5. Another more common term for this error is the *local discretisation error*.

2. The *stability* error. In one step, the local error ε_l occurs. The value of the current approximation step is being used to calculate the next step, which is expressed by the formula

   $$y_{n+1} = y_n + h\Phi(h, f)(t_n, y_n)$$

   Hence, to calculate the inclination of the line for the next step, we use a disturbed value. This issue is connected with the *stability* of the numerical method as introduced in Section 2.2.5.

3. The *convergence* error. As we have seen in Section 2.2.5, the error we are ultimately interested in is the convergence error, which can be calculated from

the consistency and stability error.

$$\|\tilde{f}(\tilde{x}) - f(x)\| = \|\tilde{f}(\tilde{x}) + \tilde{f}(x) - \tilde{f}(x) - f(x)\| \leq \|\tilde{f}(\tilde{x}) - \tilde{f}(x)\| + \|\tilde{f}(x) - f(x)\|$$

The first expression is the convergence, i.e., the distance of the overall approximation to the exact solution. The last two summands are the stability and consistency. So, the sum of both partial errors yields an estimate for the convergence error. This error is also more commonly called the *global discretisation error*.

An estimate for the global discretisation error for the Euler technique can be given as followsn [SWP12].

Lemma 14 (Global Error for Euler Approximation Method)**.** *Let $\frac{d}{dt}y(t) = f(t, y(t)), y(a) = y_0$ be an ODE that is approximated on the interval $\mathcal{I} := [a, b] \subset \mathbb{R}$ with sample step sizes h_n via the Euler method, i.e., we have an approximation sequence*

$$y_{n+1} := y_n + h_n \cdot f(t_n, y_n), t_{n+1} := t_n + h_n$$

Let L be the global Lipschitz constant for the ODE, i.e.,

$$\forall (t, y_1), (t, y_2) \in [a, b] \times \mathbb{R}^n : \|f(t, y_1) - f(t, y_2)\| \leq L \cdot \|y_1 - y_2\|$$

Let moreover be $\lambda : \mathcal{I} \to \mathbb{R}^n$ be the (unique) solution of the ODE, which is two times continuously differentiable.

Then, for the global error

$$\varepsilon_{n+1} := \|\lambda(t_{n+1}) - y_{n+1}\|$$

which is the convergence error, holds

$$\varepsilon_n \leq \frac{1}{2}(b-a)e^{(b-a)L} \max_{a \leq t \leq b} \left\| \frac{d^2}{dt^2}\lambda(t) \right\| \max_{j=0,\ldots,N-1} h_j$$

where N is the maximal amount of sample steps.

The idea behind the proof is to provide an estimate for the local discretisation error, i.e., the error that is made in one step. An estimate for the local discretisation error is found with the help of the Taylor series of the solution of the ODE. With this, the second derivative of this solution is introduced into the term. The local errors sum up to a global error, which in turn can be estimated by the exponential function as given in the formula.

Note that in Lemma 14, the consistency, i.e., the error in a single approximation step, and the stability, i.e., the error due to disturbances from local consistency errors of former approximation steps, have been considered. The ε increases with the duration of the simulation interval.

We have now a specific error estimate for the Euler method available. Generally, for single step techniques, we can define *consistency* and provide an error estimate as well.

Definition 30 (Consistency of Single Step Methods)**.** *Let $\frac{d}{dt}y(t) = f(t, y(t)), y(t_0) = y_0$ be an ODE that is approximated on the interval $\mathcal{I} := [a, b] \subset \mathbb{R}$ with sample step sizes h_n via a single step method, i.e., we have an approximation sequence*

$$y_{n+1} = y_n + h_n \cdot \Phi(h_n, f)(t_n, y_n), t_{n+1} := t_n + h_n$$

Let L be the global Lipschitz constant for the ODE, i.e.,

$$\forall (t, y_1), (t, y_2) \in [a, b] \times \mathbb{R}^n : \|f(t, y_1) - f(t, y_2)\| \leq L \cdot \|y_1 - y_2\|$$

Let moreover be $\lambda_i : \mathcal{I} \to \mathbb{R}^n$ be the (unique) solution of the ODE

$$\frac{d}{dt}y(t) = f(t, y(t)), y(t_i) = y_i$$

.

*The single step method is called **consistent** if*

$$\lim_{h \to 0} \sup_{t_i \in [a,b], y_i \in \mathbb{R}^n} \frac{\|y_{i+1} - \lambda_i(t_{i+1})\|}{h_i} = 0$$

*The method has **consistency order** $p \in \mathbb{N}$ if*

$$\exists C > 0 \exists h_0 > 0 \forall 0 < h < h_0 : \sup_{t_i \in [a,b], y_i \in \mathbb{R}^n} \|y_{i+1} - \lambda_i(t_{i+1})\| \leq C h^{p+1}$$

The constant C depends on the ODE. In Lemma 14, we have given a specific formula. The order p depends on the chosen approximation method. Euler has order 1, Runge-Kutta methods have higher orders.

Lemma 15 (Global Error of Single Step Methods)**.** *Let*

$$\lambda : \mathcal{I} \to \mathbb{R}^n$$

be the solution of the ODE

$$\frac{d}{dt}y(t) = f(t, y(t)), y(a) = y_0$$

with Lipschitz continuous f in the second argument, Lipschitz constant L. Let furthermore be

$$y_{n+1} = y_n + h \cdot \Phi(h, f)(t_n, y_n), t_{n+1} := t_n + h$$

be the sequence defined by a single step method, sample step size h. Then the following holds

1. *Is the method consistent, then*

$$\varepsilon_n = \max_{0 \leq i \leq n} \|y_i - \lambda(t_i)\| \xrightarrow{h \to 0} 0$$

2. *Has the method consistency order p, then*

$$\forall 0 < h_i < h_0 : \varepsilon_n = \max_{0 \leq i \leq n} \|y_i - \lambda(t_i)\| \leq \frac{e^{L(b-a)} - 1}{L} C h^p$$

where $C, h_0 > 0$ are the constants from the definition of the consistency order.

Proof. See for instance [SWP12] □

In this dissertation, we restrict ourselves to the Euler method for concrete calculations of the global error ε, which plays an important role for the precision of approximate bisimulation, i.e., the maximal deviation between source and target model of a given refactoring. As we have seen, Euler has a very bad error estimate. However, we want to demonstrate the principle in this dissertation. The users are free to choose other approximation techniques and consequently use better estimates as they instantiate our framework.

In the next section, we introduce MATLAB / Simulink, which we have chosen as major representative for modelling languages capable of hybrid control system development.

2.3 MATLAB/Simulink

Simulink is an addon package to MATLAB from The MathWorks, Inc. [TM17b]. It is a graph-based, signalflow-oriented modelling language and development tool. Simulink is widely used in the Model Driven Engineering process for the following reasons: Its language is intuitive to use, provides a large block library, and supports automatic code generation.

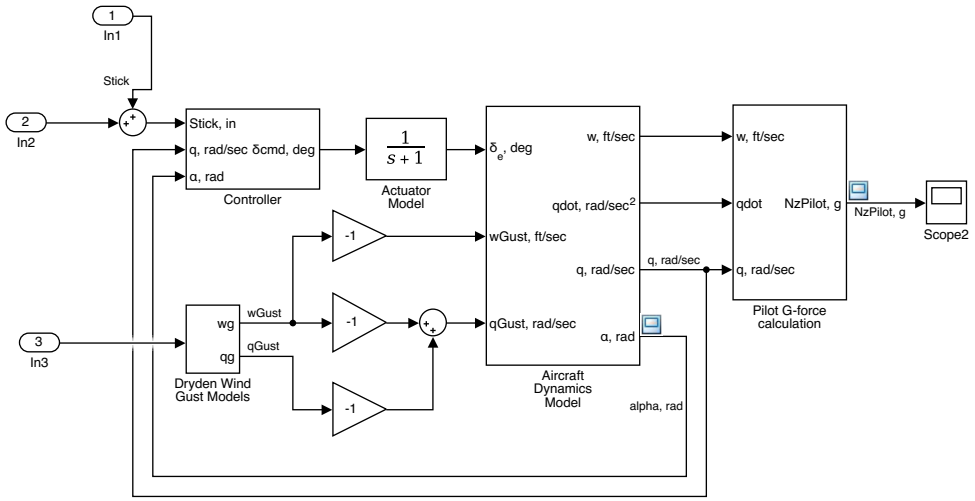

FIGURE 2.6: Simulink Model Example

2.3.1 Syntax of Simulink models

A Simulink model is a graph. The vertices are called blocks, the edges are called signal lines. An example of a Simulink model is depicted in Figure 2.6. There, a model of an F-14 longitudinal flight control is depicted. The figure illustrates blocks, signal lines, and subsystems, which contain submodels. One of the subsystems contains a model to simulate wind gusts. Another subsystem contains the controller that interprets the input coming from the stick, which is controlled by the pilot. We also use this model as one of our case studies.

Simulink distinguishes between the following block types:

- Input blocks
- Output blocks
- Function blocks or direct-feedthrough blocks, e.g. sum, product, exponential function
- Stateful blocks, e.g. Delay block, Integrator
- Subsystem blocks

Blocks can be associated with parameters. Stateful blocks like Delay blocks or Integrator blocks have an initial value. Delay blocks keep a sample step time. Figure 2.7 gives an overview over some block types. The user creates the model by dragging and dropping the blocks from the library to the respective model page.

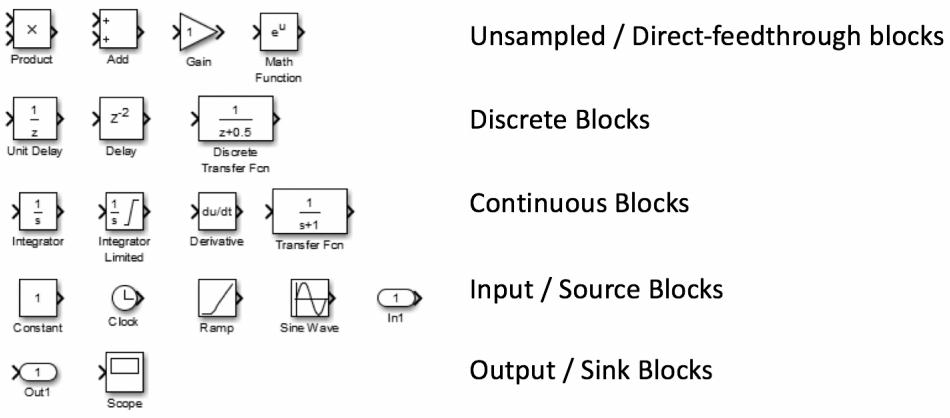

FIGURE 2.7: Examples for Simulink blocks

The blocks are connected via point and click. A submodel can be grouped to a subsystem via the context menu of the tool.

In the next subsection, we explain how Simulink models are interpreted.

2.3.2 Simulink Operational Semantics

The goal of this thesis is to obtain a methodology for formal verification of the correctness of refactorings on data-flow oriented graph-based models. We choose Simulink as representative for data-flow oriented models. However, The MathWorks, Inc. only provides informal descriptions for the semantics of Simulink [TM17b]. This precludes the rigorous analysis of Simulink models. A formalisation of the Simulink semantics is given in [BC12]. In the following, we briefly summarise the phases of the semantics of Simulink informally. After that, we introduce the formal operational semantics [BC12], which is our formal foundation.

Informal Semantics

As stated in the informal description for Simulink [TM17b], the system performs the following steps for executing its models.

1. The first phase is the *Initialisation phase*. There, the model is compiled to an executable format, linked, the parameters are initialised, and the model hierarchy is flattened.

2. The actual simulation phase is a loop. In each cycle, the outputs for each block are calculated and the states, i.e., the internal values for time-discrete and time-continuous blocks. This yields the output for the simulation in the current simulation step, i.e., at the current simulation time.

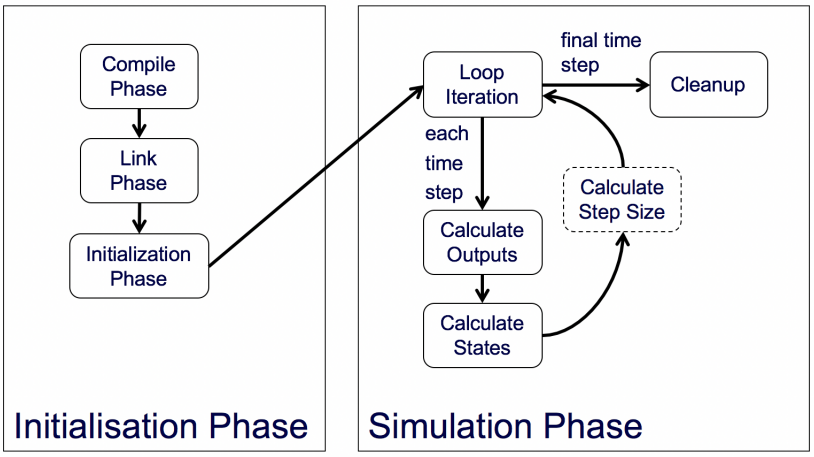

FIGURE 2.8: Overview over the evaluation steps of Simulink

3. If a variable step size solver is selected, after the calculation of the simulated values, the next step size is calculated.

4. The whole simulation ends with a cleanup.

Figure 2.8 gives an overview over the simulation steps. After this informal introduction we go over to a formally defined operational semantics.

Formal Operational Semantics

We provide a brief summary of the operational semantics [BC12], which is our formal foundation. It is assumed that all signal lines are labelled in the form l_i. Time-discrete blocks have an internal state denoted with d_i and time-continuous blocks have an internal state denoted with x_i.

Definition 31 (Semantical State of Simulink)**.** *A semantical state of Simulink is a mapping* $\sigma : V \to \mathbb{R}$ *where* $V = \{l_i | 1 \leq i \leq n_b\} \cup \{d_i | 1 \leq i \leq n_d\} \cup \{x_i | 1 \leq i \leq n_x\} \cup \{t, h\}$ *that assigns values to all output variables* l_i*, and to the internal variables* d_i *and* x_i *of discrete respectively continuous blocks. The numbers* n_b, n_d *and* n_x *express the amounts of variables for the output of blocks, discrete and continuous respectively. The distinguished variables* t *and* h *are used for the simulation time and the simulation step size. We denote the set of semantical states of Simulink with* Σ*.*

Parameters that are provided by the user are denoted by a function π. For instance, $\pi(t_0)$ expresses the simulation start time, $\pi(t_{end})$ the simulation end time, $\pi(h_0)$ the initial simulation step size.

The operational semantics is defined by providing a set of inference rules. The Simulink model is assumed to be flattened, i.e., all subsystems have been removed

and the entire model is on one level. This is consistent with the informal semantics. Simulink flattens the subsystem hierarchy prior to evaluating the model. As further preparation, a set of syntactical equations and assignments, denoted with Eq, is derived from the Simulink model. They are built by going through the model and assigning a set of equations to each block. This is done as follows. Let l_o denote the output signal and l_i the input signal annotation, or $l_1, ..., l_n$ if there are multiple incoming signals.

- For direct-feedthrough blocks, the arithmetic or logical expression or the instantiated function (in case of source blocks) is directly expressed. For example, the constant block with constant c yields $l_o = c$, the plus block with two operands yields $l_o = l_1 + l_2$.
- Time-discrete blocks keep an internal variable d, and sample steps, which are contained in a set S. The idea behind that is that time-discrete blocks are only evaluated at sample time steps in S. Between those time steps, the values remain constant. We provide further details in the inference rules.The equation set for instance for the *Unit Delay* block, which holds the input value for one sample period, is given by $l_o =_S d; d :=_S l_i, d(0) := init$, where the last equation denotes the initialisation at the simulation start.
- For time-continuous blocks, consider the internal variable given as x. Then the equation set is $l_o = x; \dot{x} = l_i; x(0) := init$.

After these preparations, the inference rules are given.

Global Simulation Rule The global simulation rule is defined as

$$\frac{\sigma(t) \geq \pi(t_{end})}{Eq, \pi \vdash \sigma \rightarrow \sigma}$$

$$\frac{\sigma(t) \leq \pi(t_{end}) \qquad Eq, \pi \vdash \sigma \xrightarrow{\text{M}} \sigma_1 \qquad Eq, \pi \vdash \sigma_1 \xrightarrow{\text{u}} \sigma_2 \qquad Eq, \pi \vdash \sigma_2 \xrightarrow{\text{s}} \sigma'}{Eq, \pi \vdash \sigma \rightarrow \sigma'}$$

The symbol Eq stands for all syntactical equations associated to blocks as defined above. The symbol π (for *parameter*) is a function that associates values to distinguished variables, such as $\pi(h)$ defines the global sample step size for fixed-step solvers. The symbol σ denotes a state, i.e., $\sigma \in \Sigma$.

The first semantic rule lets the simulation end as soon as the user-defined simulation end time has been reached. Note that the user must always define a finite execution time for an execution of a Simulink model. The second semantic rule

splits the simulation into three parts. The first transition is the $M-$ transition (*major step* transition), which basically evaluates all direct-feedthrough blocks and block outputs. This includes equations of the form $l = e$ and $l =_S e$. The second transition is the $u-$ transition (*update* transition), which evaluates the internal state of all time-discrete blocks, i.e., sampled equations of the form $d =_S l$. The third transition is the $s-$ transition (*solver* transition), which evaluates all internal states of time-continuous blocks, i.e., equations of the form $\dot{x} = l_i$, and if necessary calculates the next simulation step size. An update of the simulation step size is only necessary if the user has activated the variable step size solver option. The alternative is the fixed step solver option, which leaves the simulation step size constant.

Next, we describe the major step, update and solver transitions in detail.

Major step transitions For the major step transitions, an auxiliary $o-$ transition is introduced. This transition basically applies arithmetic or logic functions in expressions. The connection with the $o-$ transition is defined in the main major step transition rule.

$$\frac{\langle Eq(M), \sigma\rangle \xrightarrow{o} \sigma'}{Eq, \pi \vdash \sigma \xrightarrow{M} \sigma'}$$

The notation $Eq, \pi \vdash \sigma \xrightarrow{M} \sigma'$ means that one can derive from the set of equations, which is denoted by Eq that the state σ and the state σ' are in relation via the major step transition relation. The notation $\langle Eq(M), \sigma\rangle \xrightarrow{o} \sigma'$ means that the state σ is in relation with state σ' via the $o-$ transition under the determination given by the equation set $Eq(M)$. The set $Eq(M) \subseteq Eq$ is the set of equations of the forms $l = e$ and $l =_S e$, i.e., equations that determine the output of blocks.

We now provide the inference rules and give an explanation of their meaning.

$$\frac{}{\langle r, \sigma\rangle \xrightarrow{o} r} \quad \frac{}{\langle l, \sigma\rangle \xrightarrow{o} \sigma(l)} \quad \frac{\langle e_1, \sigma\rangle \xrightarrow{o} r_1 \, \langle e_2, \sigma\rangle \xrightarrow{o} r_2}{\langle e_1 \diamond e_2, \sigma\rangle \xrightarrow{o} r_1 \diamond r_2}$$

$$\frac{\langle e_1, \sigma\rangle \xrightarrow{o} true \, \langle e_2, \sigma\rangle \xrightarrow{o} r_2}{\langle if(e_1, e_2, e_3), \sigma\rangle \xrightarrow{o} r_2} \quad \frac{\langle e_1, \sigma\rangle \xrightarrow{o} false \, \langle e_3, \sigma\rangle \xrightarrow{o} r_3}{\langle if(e_1, e_2, e_3), \sigma\rangle \xrightarrow{o} r_3} \quad \frac{\langle e, \sigma\rangle \xrightarrow{o} r}{\langle l := e, \sigma\rangle \xrightarrow{o} \sigma[l \leftarrow r]}$$

The first set of inference rules describes how expressions are evaluated. The symbol r in the first rule is a constant. This evaluates to its value. The second rule states that the value of a variable symbol is determined by its value in the state.The third rule is the application of arithmetic or logical functions, e.g. addition, product, exponential function, and, or. The symbol $\diamond$ refers to the respective operation. In

the second line, the first two rules define the behaviour for switch blocks, which are represented by the $if(e_1, e_2, e_3)$ symbol in the preparation. The third rule in the second line defines how an assignment is treated. The value replaces the value of the respective variable in the state. The notation $\sigma[l \leftarrow r]$ is defined as

$$\sigma[l \leftarrow r](x) := \begin{cases} r & \text{if } x = l \\ \sigma(x) & \text{otherwise} \end{cases}$$

The next set of rules defines how sampled outputs are calculated, i.e., it deals with assignments of the form $l :=_S e$. Such equations occur at time-discrete blocks, e.g. the unit delay block. They have two kinds of variables - internal (denoted as d) and output variables (denoted as l). The update of the internal variables is defined in the $u-$ transitions, which we introduce in the next paragraph. The update of the (sampled) output variables are part of the $M-$ transitions.

$$\frac{\sigma(t) \notin S}{\langle l :=_S e, \sigma\rangle \xrightarrow{o} \sigma} \qquad \frac{\sigma(t) \in S \quad \langle e, \sigma\rangle \xrightarrow{o} r}{\langle l :=_S e, \sigma\rangle \xrightarrow{o} \sigma[l \leftarrow r]}$$

The first rule states that the value of the output remains constant, i.e., the state does not change if the sample step time has not been reached. The state σ keeps for variable t the current simulation time. The second rule deals with precisely this case where the sample time has been reached. In this case, the expression is evaluated and the respective value replaces the value in the state for variable l.

In the next paragraph, we explain how the internal variables for time-discrete blocks are updated.

Update transitions The update transitions deal with the evaluation of internal variables of time-discrete blocks. The respective assignments are of the form $d :=_S l$. The variable l is in this case associated with the incoming signal line of the respective block.

$$\frac{d :=_S l \in Eq \quad \sigma(t) \notin S}{Eq, \pi \vdash \sigma \xrightarrow{u} \sigma} \qquad \frac{d :=_S l \in Eq \quad \sigma(t) \in S}{Eq, \pi \vdash \sigma \xrightarrow{u} \sigma[d \leftarrow \sigma(l)]}$$

The first rule states that the value of the internal variable remains constant, i.e., the state does not change if the sample time has not been reached. If the sample time has been reached, the second rule applies. In this case, the value of the internal variable is replaced by the value of the incoming signal, which is represented by the evaluation of the state for the variable l.

In the next paragraph, we describe the evaluation of time-continuous blocks.

Solver transitions Note that the output variables are calculated in the major step transition phase. The assignment defining the output for time-continuous blocks of the form $l_{out} := x$, where x is the internal variable of the respective block.

We provide the solver transition rule from the operational semantics.

$$\frac{Eq, \pi \vdash \sigma \xrightarrow{i} \sigma_{so} \quad Eq, \pi \vdash \sigma_{so} \xrightarrow{zc} \sigma zc \quad Eq, \pi \vdash \sigma_{zc} \xrightarrow{h} \sigma'}{Eq, \pi \vdash \sigma \xrightarrow{s} \sigma'}$$

This rule splits the solver transition into three parts. Firstly, the integrator transition ($i-$ transition) is executed, which calculates the next approximation step for time-continuous blocks. Note that although the blocks are called *time-continuous*, their evaluation in the simulation semantics, which is defined in the operational semantics, is not continuous in the mathematical sense. It has a finite step size, which is generally smaller than for time-discrete blocks, and may be of variable step size. However, its size is a finite number. The evaluation of time-continuous blocks is therefore always an approximation. This results in the challenges we deal with in this thesis. During the solver transition, the simulation step size can be modified. However, this only applies if the user has activated the variable step size simulation option.

The second part is the *zero-crossing detection* ($zc-$ transition). This is required if blocks impacting the control flow, e.g. switch blocks, are present in the model. Such blocks feed through either one signal or another depending on the crossing of the defined threshold by the control signal. If the user activated variable step size simulation, the system tries to approximate the location of the threshold crossing up to a user-defined precision.

The third part constitutes the calculation of the next simulation step size. During the integrator transition, the simulation step size can already be modified. The zero-crossing detection may modify the simulation step size again. The $h-$ transition takes the minimum of both.

Note that parts of the first (integrator transition), the second (zero-crossing detection) and third (next simulation step size calculation) parts of the rule only apply if variable step size calculation has been activated. Otherwise, the solver transition consists only of the approximation parts of the integration transition. Since in this thesis, we only consider fixed-step simulation (variable step simulation is part of future work), we skip the semantical rules for zero-crossing detection and calculation of the next simulation step size.

For fixed-step simulations, the solver transition therefore consists only of the integrator transition. The integrator transition is the application of the chosen approximation method, e.g. Euler, with the fixed simulation step size. In essence,

the solver transition reduces to the following in the context of this thesis.

$$\frac{Eq, \pi \vdash \sigma \xrightarrow{i} \sigma'}{Eq, \pi \vdash \sigma \xrightarrow{s} \sigma'}$$

In this rule, the integrator transition is the application of the chosen approximation technique.

Note that just like the update and major step transitions, the integrator transitions are applied for all time-continuous blocks *simultaneously*, i.e., the operational semantics is a synchronous semantics.

2.4 Summary

In this chapter, we have provided the preliminaries that we use in this thesis. We have started with a brief introduction to model refactorings and how they embed into Model Driven Engineering, which is a common development approach for embedded systems.

We have proceeded with mathematical preliminaries. In particular, we have introduced some topological concepts, which are the basis of many aspects of mathematical analysis and help to understand the concepts time-discreteness and time-continuity. We then have introduced difference equations, sequences, and most important properties of sequences. We use difference equations and sequences to describe the behaviour of time-discrete models in this thesis. We then have introduced differentiation of functions, important properties of functions and sequences of functions, and differential equations. The concepts of this section are of concern for time-continuous models. We have introduced Laplace transform and z-Transform. We use these transformations to obtain algebraic expressions from ODEs and difference equations. These expressions can automatically be checked for symbolic equivalence with the help of a Computer Algebra System in our automation approach. We have closed the mathematical preliminaries with an introduction of concepts from numerical mathematics. We use these concepts to obtain estimates for the maximal difference between source and refactored target model due to the approximation induced by the semantics.

We have closed this chapter with a discussion of the informal and formal semantics for Simulink. The formal operational semantics [BC12], which we have presented in this chapter, is the formal foundation of our approach.

In the next chapter, we discuss related work.

Chapter 3

Related Work

Hybrid systems can be modelled in control flow-oriented languages, and data flow-oriented or signal flow-oriented languages. The most prominent control flow-oriented hybrid systems modelling language are hybrid automata. The most important data flow-oriented or signal-oriented modelling languages are MATLAB/ Simulink, Modelica, and LabView. For both types of modelling languages, a broad spectrum of work related with hybrid systems modelling, analysis and verification, and verification of behavioural equivalence exists. Additionally, various articles concerning model refactorings on signal flow-oriented modelling languages, particularly on Simulink, exist. However, to the best of our knowledge, there is no known approach that concerns with formal verification of behavioural equivalence on data flow-oriented languages such as Simulink available. Approaches concerning behavioural equivalence do not account for the approximate nature that industrial signal flow-oriented languages such as Simulink exhibit.

In this chapter, we give an overview over related work and distinguish our approach from it.

3.1 Hybrid Systems Verification and Analysis

In this section, we discuss approaches related with verification and analysis of hybrid systems modelling languages. We start with control-flow oriented hybrid systems modelling languages and proceed with data flow-oriented languages. For both types of modelling languages, we discuss verification approaches for properties of models, and verification approaches to show behavioural equivalence.

3.1.1 Control Flow-Oriented Hybrid Systems Modelling Languages

There are various languages to model hybrid systems in a control flow-oriented manner. The classical and most prominent representative are hybrid automata

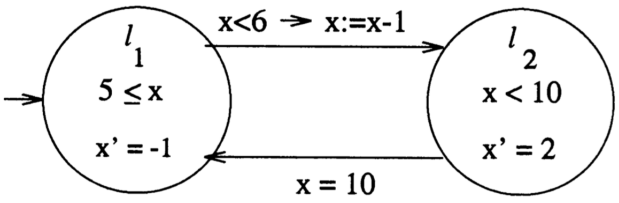

FIGURE 3.1: An example for a hybrid automaton

[ACHH93]. An operational semantics for hybrid automata is defined e.g. in [LZ05], a denotational semantics in [EP06].

Figure 3.1 depicts a simple example of a hybrid automaton taken from [ACHH93]. In timed and hybrid automata, variables can be defined. The semantic state is a pair consisting of the current location and the assignment storing the values for each variable. There are two possible transitions: timed transitions (timed automata) or continuous change (hybrid automata), and discrete changes. Timed transitions are changes of the assignments, i.e., the variables' values change. For timed automata this means a strictly monotonic increase of the real values assigned to variables. For hybrid automata, this means a continuous evolvement of real values assigned to variables that follow the trajectories of differential equations given in the location that is currently active in the semantical state. According to [MMP91], it is possible to define both time-discrete and time-continuous variables. Time-discrete variables keep their values constant in a defined interval, while time-continuous variables define an arbitrary change of values over time - as long as it follows a continuous function. Discrete transitions are changes of the location. This is possible if the property of a guard associated with a transition originating from the current location is fulfilled. The focus for this kind of modelling is therefore at the definition of value evolvement over time and control flow rather than defining signal flow. We therefore speak of control flow-oriented hybrid modelling languages, among which hybrid automata are the major representatives.

Verification of properties for general hybrid systems quickly becomes undecidable, e.g. the question if a run exists is undecidable [ACHH93, HKPV95]. Therefore, hybrid systems are restricted to be able to verify some properties for constrained systems. An example for such constrained systems are linear hybrid systems [SRKC00, VCSS03, HHWT97], i.e., systems that are defined by linear differential equations.

Alternative representations of hybrid systems, i.e., apart from hybrid automata, describe the behaviour directly as differential equation systems. This can be enriched by conditions for discrete jumps, as for example in switched systems [Bra98, PP03].

Switched systems define a set of differential equations that apply if a given condition is met. This is a concept analogous to the discrete transitions that change the trajectory defined as differential equation in one location to a trajectory defined by a different differential equation in the subsequent location. The trajectories between switches in such systems are smooth, i.e., arbitrariliy often differentiable. There exist other forms and variations of hybrid systems, e.g. a subform of switched systems are sampled-data hybrid systems [SK00b], which force the discrete jumps to occur at defined sampling times. This is a restriction to guards at transitions. Moreover, an important variation is the representation of hybrid systems as hybrid programs, where Platzer defined hybrid programs together with a Differential Dynamic Logic [Pla08], a hybrid analogon to Timed Logics.

Property Checking

There exists a very broad spectrum for hybrid system analysis and verification approaches. One major domain is a calculation of an over-approximation for reachability sets, i.e., subsets of the state space where the trajectories resulting from executions of the system are located. Examples for such over-approximations are calculations of flow pipe polyhedra [CK03] and Taylor approximations [LT05]. Other approaches follow a symbolic analysis [AHH96], or use abstractions [AHLP00, Tiw08], i.e., reduce the state space by defining a less complex model that presumes the properties of interest. Moreover, symbolic simulation has been discussed in [DM07, JFA+07]. Such approaches bundle inputs by using symbolic analysis techniques to identify such simulation traces that do not differ with respect to the property that is desired to be verified.

Besides reachability analyses, stability and robustness analyses are of special interest for hybrid systems, e.g. in [GST12, Laz06, PL96], or for switched hybrid systems in [PP03, Bra98]. Stability basically means that a variation of initial values does not lead to diverging behaviour. Either a global threshold, i.e., a maximal distance of the disturbed solution in comparison to a distinguished solution, the equilibrium or attractor can be given, or disturbed solutions converge against the equilibrium. The downside of these approaches is that they require a Lyapunov function [Tes12]. Such a function guarantees stability. It is important especially for non-linear systems. While the check for a given function to establish if it is Lyapunov is a simple task, an automated search for a suitable Lyapunov function is impossible. In our approach, we do not consider stability of the differential equations. Our estimate for the maximal disturbance, which in turn yields the precision ε of the approximate bisimulation, is possible without constraints.

There are various tools available that support automated verification. HyTech [HHWT97, AHH96] was the first model checker for reachability analysis on hybrid systems. It uses polyhedra over-approximation. The models are restricted to linear hybrid systems. The tools CheckMate [SRKC00, SK00a, CK99] and d/dt [ABDM00] also calculate over-approximations of reachable sets using polyhedra. SpaceEx [FLGD+11] is an approach that scales better than the previous approaches. It uses other over-approximation techniques than polyhedra for linear hybrid systems. Another important tool, which is a theorem prover capable of machine-assisted verification of hybrid programs and Differential Dynamic Logic is KeYmaera [QML+15, PQ08].

Equivalence Checking

An important part of hybrid systems analysis concerns the verification of behavioural equivalence. To this end, the equivalence notion of *approximate bisimulation* has been introduced [GP11]. We adapt this concept in our approach to data flow-oriented languages, particulary Simulink, to cope with the approximate nature of such models. There are various approaches that apply the concept of approximate bisimulation and describe verification of behavioural equivalence for hybrid systems. Examples are [GP07a, Gir05] for constrained linear systems, and [PGT08, GP05] for non-linear systems. In [GP07b], an approach has been proposed to calculate a precision ε for approximate bisimulation. It requires a Lyapunov-like function [Tes12] for the estimate. Another approach for conformance in hybrid models is [AMF14]. There, the authors introduce a new notion of approximately equivalent behaviour, which takes time step distances besides the distances of values into account. Furthermore, the authors calculate the conformance, i.e., approximately equivalent behaviour, based on running tests, e.g. between simulated model and generated code.

All these approaches assume an exact evaluation of the differential equations. They therefore do not account for the approximate nature we consider for industrial signal flow-oriented languages.

3.1.2 Data Flow-Oriented Hybrid Systems Modelling Languages

Important representatives for data flow-oriented languages that are widely used are Simulink, Modelica, and LabView.

Simulink is a data flow-oriented language for hybrid control systems. It is an extension of MATLAB. For Simulink's semantics [TM17b] only informal descriptions exist. While it is sufficient to work with Simulink from a user perspective, they are

insufficient to provide a formal foundation for formal verification. There therefore have been various approaches to formalise the semantics and perform verification on Simulink models. In most cases, a formal semantics has been achieved by mapping subsets of Simulink to some well-established formal languages. For example, in [HRB13], Simulink models are mapped to UCLID. In [TSCC05], this is done with the synchronous data flow language LUSTRE. In [Bos11], the semantics of Simulink is described via synchronous data flow graphs. All these approaches only cope with time-discrete models. In [BCC+17], the authors translate hybrid Simulink models to Zélus, a synchronous language that conservatively extends Lustre with ODEs to program systems that mix time-discrete and time-continuous signals. In [ASK04], Simulink models are translated to hybrid automata. This enables further investigation of hybrid automata semantics, e.g., in [EP06] or [LZ05]. The same is performed in [MMBC11]. There, the authors use the tool HyLink to translate Simulink models to hybrid automata. However, the approximate nature of Simulink is no concern in these approaches.

Modelica is an equation-based, data flow-oriented or signal flow-oriented language for modelling hybrid systems. The models are defined textually, in a data flow-oriented way. Modelica also has a data flow-oriented visualisation. As it is the case with Simulink, Modelica has only an informal description [Ass17, FE98]. In [Å98], a first approach has been proposed to provide a formal semantics for parts of Modelica. There, Modelica structures are translated to a formal language called Flat Modelica. Another approach is given by [FTCW16]. In this article, the authors develop a theory of hybrid relations inspired by Hybrid CSP [LLQ+10] and Duration Calculus [CRH93]. With this, semantics for a subset of Modelica can be expressed.

LabView [WA10, EVZH07] is a graphical programming language for data flow-oriented hybrid control models. In [MS98], a real-time semantics for a basic subset of LabView has been proposed.

In most of these approaches, only parts of the languages have been considered, e.g. [HRB13, TSCC05] considers only time-discrete Simulink models. While [MMBC11, ASK04] opens the possibility to use tools for hybrid systems for verification, the approximate nature of data flow-oriented languages is not taken into account. [BCC+17] expresses hybrid Simulink models in terms of a functional language. However, they also do not consider approximate executions. The first and only known approach to describe Simulink's semantics comprehensively and formally as operational semantics has been conducted in [BC12]. We use this as our formal foundation. Based on this, we construct a denotational semantics.

Property Checking

Above, we have provided an overview over existing approaches to formalise semantics of data flow-oriented modelling languages. Many of these approaches aim at property checking and define a formal semantics as prerequisite for formal verification. Hence, many of the approaches from above apply as well to property checking. For instance, in [HRB13], the SMT (satisfiability modulo theories) solver UCLID is used for verification of properties on Simulink models. In [TSCC05], this is done with the synchronous data flow language LUSTRE. In [Bos11], an approach for contract based verification is presented.

In [RG14], the authors use Boogie, a verification framework developed at Microsoft Research, for verification of time-discrete Simulink models. A different approach for time-discrete Simulink models is [AKRS08]. There, the authors symbolically execute time-discrete Simulink models in order to obtain a set of initial states such that the execution of Simulink starting from those states leads to behaviour equivalent to a given initial state. Simulink also comes with a tool for verification itself: the Simulink Design Verifier [HDE+08]. The idea is to provide additional blocks for Simulink allowing to check various properties for signals, and to check for compliance with modelling guidelines.

In [KKR09], an approach for formal verification of properties of LabView models with the ACL2 Theorem Prover [KM97] has been proposed. To this end, annotated LabView programs are translated to ACL2.

All these approaches do not consider behavioural equivalence of data flow-oriented models.

Equivalence Checking

There are various approaches considering verification of behavioural equivalence of Simulink models. In [MSUY13, SSD12, CMKSP10, RS09], approaches are presented to verify equivalence between Simulink models and generated code. They distinguish themselves from our approach by comparing model to code rather than comparing two models. The same applies to [SDS15, SSD12], where the authors compare Simulink with low level implementations. In [HKAS14], an approach to verify constrained equivalence for Simulink models is desribed. The authors translate Simulink models to the formal language PVS [ORS92] and use this system to verify behavioural equivalence. However, this approach is also only capable of dealing with simple directed models without feedback loops. Moreover, the approximate nature of Simulink is not taken into account.

To the best of our knowledge, there are no known approaches for Modelica or LabView concerning behavioural equivalence.

3.2 Model Refactorings for Data Flow-Oriented Languages

For the design and application of refactorings in Simulink, several approaches exist. In [Mat14], a taxonomy of model mutations is defined. The author's idea is to inject various small modifications of the model to enable enhanced clone detection. Clone detection, e.g. described for Simulink in [DHJ+10], is a major source for model refactorings. Model clones in a narrow sense are subgraphs of the model that occur multiple times in the model. In a broader sense, a model clone is a subsystem occuring multiple times in the overall model, and exhibiting the same or equivalent behaviour. This means, the clones can be of different appearance from graphical or syntactical perspective, but have an equivalent behaviour. An induced model refactoring in this case is to delete all occurences of the found clones, establish a new subsystem with one of the representative of the clones, and correctly reconnect the newly established subsystem. In [ABSH11], this is further developed yielding a normalisation of Simulink models. Simulink offers model clone detection as part of their product Simulink Check [TM17a]. However, all these approaches are limited to syntactical or very simple algebraic refactorings. In particular, they support renaming of blocks, assembling of model parts into subsystems, and changes of multiplications of constants or similar simple arithmetic changes. They do not provide an approach for formal verification of behavioural equivalence.

In [TWD13], the authors present Simulink refactorings that allow subsystem and signal shifting in arbitrary layers. This means, blocks can be assembled to subsystems, signals in busses. This operation can be reversed: A signal can be extracted from busses, blocks can be shifted in the subsystem level hierarchy as well. The approach ensures a correct reconnection in the process. The approach is limited to syntactical or very simple algebraic refactorings as well.

For Modelica refactorings, only few approaches exist. In [FPNB08], ideas for text-based refactorings for Modelica are proposed. The aim of the presented approach is to provide a pretty-printer for Modelica that keeps remarks and formatting in the refactoring process. The refactorings are only of syntactical nature. In [HNNA09], refactorings in the form of renaming are supported.

In [SLZ08], a general approach for refactoring on Dataflow Visual Programming Languages (DFVPLs) is presented and applied to LabView. There, the authors present a framework for data flow-oriented languages. The supported refactorings are adding and removel of parameters, change order of parameters, renaming,

function or subsystem removal or extraction. These refactorings therefore also mainly focus on syntactical refactorings.

In summary, there are various approaches for refactoring of data flow-oriented languages. However, they mainly focus on syntactical refactorings. Semantical refactorings, e.g. the replacement of a subsystem represented by a differential equation by a subsystem representing its analytical solution, are not considered in these approaches. Moreover, they do not provide formal verification of behavioural equivalence.

3.3 Summary

There exists a broad spectrum of approaches that is related to our thesis. First of all, there is the wide area of control flow-oriented hybrid system modelling languages, of which hybrid automata is the most prominent representative. However, these approaches assume an exact evaluation of time-continuous model parts. In this thesis, we aim for behavioural equivalence on industrially relevant data flow-oriented or signal-oriented languages. Such languages are executed on machines with finite execution time. Their semantics therefore involves approximations, which control flow-oriented hybrid system modelling approaches do not account for.

Secondly, there are various approaches concerning data flow-oriented or signal flow-oriented languages. The most prominent representative is Simulink, which we have choosen in our approach to demonstrate our concepts. They lack formal semantics, which is typically overcome by translating subsets of the languages to formal languages. [BC12] provides a formal operational semantics for Simulink, which is comprehensive, and accounts for the approximate nature of industrial signal-oriented languages, particularly Simulink. We therefore use this approach as our formal foundation. Approaches for verification of data flow-oriented hybrid modelling languages do either not concern with behavioural equivalence or do not account for the approximate nature that industrial signal-oriented languages exhibit.

Thirdly, there are various approaches considering refactorings of data flow-oriented hybrid models. However, firstly they aim mainly at syntactical refactorings such as subsystem merge, or simple arithmetic changes in models without feedback loops. Secondly, they do not consider verification of behavioural equivalence.

To the best of our knowledge, there is no approach available yet that allows for the automated verification of refactorings for hybrid data flow-oriented or signal-oriented models.

Chapter 4

Formal Verification of Behavioural Equivalence of Hybrid Control System Models

Embedded systems are often employed in safety-critical environments. Models tend to become very complex in the model-driven engineering process and often violate modelling guidelines. Refactorings are very common to reduce complexity and establish compliance with guidelines. However, they are often performed manually, e.g. as in [TWD13]. In this process, it is likely to introduce erroneous behaviour. To overcome these problems, we propose an approach to verify the results of the transformation, i.e., we define a methodology that enables automated verification of behavioural conformance of a hybrid control system model and its transformed counterpart.

Our approach is twofold. Firstly, we provide a theoretically well-founded methodology for the formal verification of behavioural equivalence of dataflow-oriented hybrid system models. Secondly, we provide an approach for checking behavioural conformance for a linear subset of hybrid control systems. This approach enables largely automated proofs. We achieve this by automatically generating proof obligations for data flow-oriented hybrid control system models, and by checking symbolic equivalence by using a Computer Algebra System (CAS). We have published our methodology and automation approach in [SHGG18].

In this chapter, we firstly discuss limitations and assumptions of our approach. Then, we provide an overview of our methodology. Finally, we briefly introduce of our approach for a semi-automated equivalence checker for a subset of data flow-oriented hybrid control models.

4.1 Limitations and Assumptions

We split our limitations and assumptions to those that apply to our methodology and those that apply to our automation approach.

4.1.1 Methodology Limitations

The following limitations and assumptions apply to our methodology.

1. We do not consider MATLAB S-Function blocks, libraries, and external code.
2. While we consider virtual subsystems, we do not consider atomic subsystems.
3. We do not support algebraic loops, i.e., each cycle must contain at least one stateful block.
4. Our approach is only able to verify correctness of refactorings that keep the control flow, i.e., control flow elements are not reduced.
5. Time-discrete and time-continuous blocks are not used in the same feedback loop.
6. While we do consider approximation errors resulting from numerical solving of ODEs, we do not consider approximation errors resulting from arithmetical operations on floating point values.
7. We assume fixed simulation step size.

The first three limitations concern the blocks that we currently support. We do not support external code or MATLAB code. In this thesis, we focus on data flow-oriented languages and do not consider textual programming languages. We refer atomic subsystems to future work. Algebraic loops do not play an important role in industrially relevant applications. The idea for the usage of feedback loops is mainly to model reactive behaviour, which requires a stateful block within the loop. The fourth limitation concerns with completeness of our approach. Our approach declines refactorings altering the control flow - even though the refactoring may be correct. An example for a refactoring that would be declined although exhibiting equivalent behaviour would be if one data flow path is never reached because the control flow condition is never fulfilled. In this case, the particular data flow path and the control flow block could be removed, which would be a correct refactoring. However, our approach is not able to verify this automatically, i.e., our approach is not complete. It should be noted that due to Rice's theorem [HMU01], it is not possible to obtain an automated approach that decides if two arbitrary data flow

models behave equivalently. The fifth contribution limits supported models. Our theoretical methodology in principle also supports models where time-discrete and time-continuous parts occur together in the same feedback loop. However, models containing both time-discrete and time-continuous parts, mathematically represent delay differential equations. Such equations generally only allow interval-wise solutions, which limits the possibilities substantially for refactorings. Since we aim at verifying correctness of relevant refactorings, our restriction is reasonable. However, we discuss cases where time-discrete and time-continuous parts occur together in a feedback loop in Chapter 6 and point into directions for future work to cover this aspect as well. The sixth constraint is conform with the operational semantics [BC12], our formal foundation. In this thesis, we focus on numerical approximation errors from discretisation of time-continuous model parts rather than floating point arithmetic errors. We have not considered variable step size semantics in this thesis. However, by considering the additional semantical rules for variable step size solving, our methodology can be extended. We leave this for future work.

There are some additional constraints whose purpose is mainly to simplify the notation.

1. We assume that all stateful time-discrete blocks (if present) are sampled with the same step size.

2. The simulation start is $t_0 \geq 0$.

The first limitation simplifies the notation. If we allowed varying sample step sizes among time-discrete blocks, sequences would not suffice to describe the behaviour of time-discrete models. However, our methodolgy can be extended to cope with differing sample step sizes among time-discrete blocks. The second assumption is a prerequisite to apply the Laplace transformation in our automation approach. However, it is always possible to translate the simulation start time to 0 if this is not directly the case.

4.1.2 Automation Approach Limitations

In order to obtain a largely automated solution for behavioural equivalence verification, our automation approach is limited to a subset of Simulink models. In particular, we apply the following additional constraints to our automation approach and our implementation.

1. We only consider Switch blocks as elements impacting the control flow.

2. We only consider Delay blocks and Discrete Transfer Function blocks as representatives for time-discrete stateful blocks, and Integrator and Transfer Function blocks as representatives for time-continuous stateful blocks.

3. All differential or difference equations that are extracted from the models must be linear for the automation approach. Input signals are connected to cyclic submodels via Sum blocks only.

The main focus of our automation approach and implementation is to 1) provide a semi-automation, and 2) demonstrate the applicability on a prototypical, conceptual level. The first constraint limits the control flow elements. Control flow elements in data flow-oriented languages control which data is fed through. This is to a certain extend comparable to an if-then-else-statement of textual programming languages. Hence, on a conceptual level, the basic Switch block incorporates the main attributes of a control flow element. The second constraint limits the stateful blocks. However, conceptually, the blocks we support exhibit the relevant time-discrete and time-continuous behaviour. Moreover, even with the restriction to these blocks, our automation approach allows to verify correctness of many industrially relevant refactorings. The third constraint is a tribute to obtain a largely automated approach. Non-linear equations required a case by case consideration in many cases, which would lead to a mechanised approach with many manual interactions rather than semi-automation.

We are confident that our approach can be extended to cope with all limitations in future work.

4.2 Theoretical Methodology

Figure 4.1 provides an overview over our methodology for verification of behavioural conformance of hybrid control system models. We consider three levels. Firstly, on the syntactical level, we have the Simulink models we wish to compare. These models are semantically interpreted in the second level, the operational semantical level as depicted in Figure 4.1. The third level is the denotational level. It describes the behaviour of the models in a denotational style. In Figure 4.1, our main contributions are depicted in numbered boxes. The numbers refer to the order in which we describe the contributions in this section.

Automated Extraction of the Abstract Representation As depicted in Figure 4.1, the Simulink models are interpreted by an operational semantics. We use the semantics described in [BC12] as formal foundation. It defines a transition system.

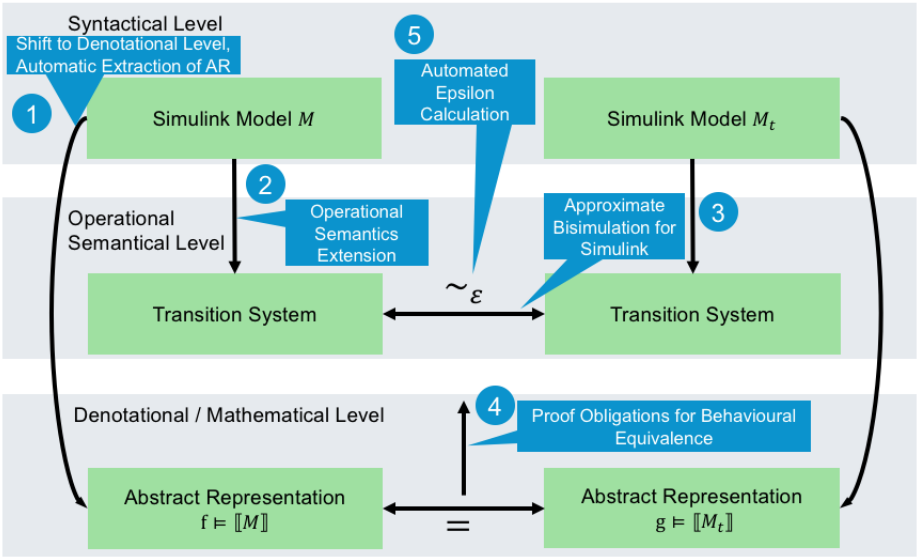

FIGURE 4.1: Overview over the Equivalence Proof Methodology

The transitions between states are formally described by inference rules. We have provided a summary of the approach [BC12] in Chapter 2. However, the operational semantics is not suitable for an automated verification of behavioural conformance if semantical refactorings, such as replacement of a differential or difference equation by its analytic solution, are involved. The reason is the following: To prove behavioural conformance, we compare the trajectories of the executions of the models. The operational semantics describes the behaviour step-wise rather than considering the trajectories as a whole. We have therefore defined a formal semantics of Simulink on a denotational level, which we call *Abstract Representation (AR)*. We denote the AR of the model M as $[\![M]\!]$. It consists of equations describing the behaviour of the model. In case of time-continuous models, the AR comprises differential equations, in case of time-discrete models, the AR comprises difference equations. To cope with control flow elements, we define a set of models that represents all possible data flows. We call these *data flow models*. A data flow model represents an execution of the Simulink model if the conjunction of conditions given as parameter at control flow elements is fulfilled. For each data flow model, the AR is extracted. The union of ARs of all data flow models together with the conjunction of conditions at control flow elements properly represents the semantics of the Simulink model.

The solutions or interpretations of the AR, denoted as $f \models [\![M]\!]$, represent the trajectories of the execution of the models in an exact, mathematical sense. For time-continuous models, the solutions are time-value functions, for time-discrete models, the solutions are sequences, which in turn represent interval-wise constant

functions. Our AR is sound with regard to the operational semantics in the following sense: Consider a sequence of operationally semantical executions with decreasing simulation step size. Then, this sequence of trajectories at the points of observations converges uniformly to the mathematical interpretation $f \models [\![M]\!]$, i.e., the function f solving the AR $[\![M]\!]$. In our first main contribution, we define the AR, prove its soundness, and describe how the AR can be automatically extracted from the models. We have published this in [SHGG15, SHGG16b] and describe it in Chapter 5 in this thesis.

Operational Semantics Extension In order to be able to cope with a subset of Simulink models that is relevant for several industrial applications, we extend the operational semantics [BC12]. Our extension comprises semantics of Transfer Function blocks and logical blocks. Particularly Transfer Function blocks play an important role in many applications. We extend the operational semantics by providing alternative expressions for the blocks that we add that exhibit equivalent behaviour in the operational semantics. The extension of the semantics is then achieved by reducing the semantics of those blocks to semantic rules that are already covered by the operational semantics. Additionally, we demonstrate how stateful blocks apart from those we have described in Section 4.1 can be expressed by semantically equivalent subsystems we already support. With that, we demonstrate how our approach can be extended to cope with other blocks as well. This forms our second contribution.

Approximate Bisimulation for Simulink The question our approach answers can be found on the operational semantical level and can be stated as 'Do the transition systems conform to each other?' As described in Chapter 1, we cannot always conclude that the semantical executions of both models yield exactly the same values. The reason for this is basically that although time-continuous functions map values to *uncountably* many time steps, concrete solutions of data flow-oriented languages such as Simulink are only able to calculate values for *finitely* many time steps. Consequently, these tools need to *approximate* operations such as approximate integration. However, we can calculate a maximal distance for the values at the points of observation. This maximal distance depends on some parameters determined by the model, the chosen approximation method, the sample step size, and the interval length of the execution of the Simulink model. It increases with the simulation interval length. Hence, we provide this distance for a specific execution with a specific interval length. To obtain a formally well-founded equivalence notion that accounts for approximal distances rather than exact values, we adapt

the approximate bisimulation [GP11] to Simulink. This is our third contribution. We have published this in [SHGG15, SHGG16b] and describe it in Chapter 6 of this thesis.

Automated Derivation of Proof Obligations To prove aproximately equivalent behaviour of Simulink models, we focus on the denotational level and show exact equality for the interpretations of the respective ARs. The central theorem, which forms our third contribution, can chiefly be reduced to the following statement: Consider the interpretations of the ARs

$$f, g : \mathcal{I} \to \mathbb{R}, f \models [\![M]\!], g \models [\![M_t]\!]$$

Here, $\mathcal{I}$ stands for the simulation interval, M, M_t for the models we wish to compare. If we can show

$$\forall t \in \mathcal{I} : f(t) = g(t)$$

then we can conclude

$$M \sim_\varepsilon M_t$$

The latter formula stands for approximate bisimulation with precision ε between M and M_t. In our contribution, we provide various proof obligations that need to be checked in the process of automated verification in order to draw the conclusion for approximate bisimulation. The main part is a symbolic equivalence check to show that the interpreting functions of the ARs of source and target model in all data flow models are the same. We have published this in [SHGG15, SHGG16b]. Another set of proof obligations concerns with conditions at control flow elements. They ensure that no time-delayed behaviour due to different switching in source and target model occurs, or identify time intervals in which such behaviour may occur. We have published this in [SHGG16a]. We describe this fourth contribution in Chapter 6 of this thesis.

Automated Epsilon Calculation Approximate bisimulation is a notion to express behavioural equivalence within an ε environment, i.e., the values the execution yields vary around each other with a maximal distance of ε. The case $\varepsilon = 0$ implies a coincidence of approximate bisimulation and traditional bisimulation. We show that the case $\varepsilon > 0$ occurs if a refactoring is the (partial) replacement of an ODE by its analytical solution. Moreover, we provide means to calculate ε. In general, the calculation of ε depends on the chosen approximation technique, the simulation step size, an estimate of the upper bound of the second derivative of the solution of

the AR, the length of the simulation interval, and the Lipschitz constant, which is a constant being inherent with the ODE of the AR. We describe how each of these parameters can automatically be obtained from the AR, and how they are used to automatically calculate the ε. If $\varepsilon > 0$, the value of ε increases with the simulation interval length. If we want to explicitly emphasise thie dependency, we denote this by $\varepsilon(\#\mathcal{I})$, where $\#\mathcal{I}$ denotes the length of the simulation interval $\mathcal{I}$.

We have published this in [SHGG15, SHGG16b, SHGG16a] and describe this fifth contribution in Chapter 6 of this thesis.

The contributions depicted in Figure 4.1 form our methodology. We provide a semi-automated approach for verification of behavioural equivalence for a subset of Simulink models whose AR only contains linear differential or difference equations. We describe the elements of our semi-automation in the next section.

4.3 Automation of our Equivalence Proof Methodology

Our automation approach applies our methodology. The general idea is as follows.

1. We automate the extration of the AR. Prior to that, the data flow models are being identified, and the model is partitioned into subcomponents, whose equivalence can be verified separately.

2. We symbolically verify equivalence of the ARs for all data flow models with the help of a Computer Algebra System (CAS). In addition, we verify the supplemental proof obligations, e.g. the conditions concerning time-delayed behaviour at control flow elements, to conclude approximate bisimulation according to our methodology.

3. We calculate the local precision ε_l of the approximate bisimulation with the help of a CAS, and we propagate the local ε_l to obtain a global precision ε_g.

Figure 4.2 provides an overview over our automation approach. The arrows depict the direction of the process. It starts with two Simulink models we wish to compare at the top of Figure 4.2.

Data Flow Models In the first step, we calculate all possible data flows in the model. This addresses the semantics of the control flow blocks. This step yields a set of Simulink models in which the Switch blocks have been eliminated. Each data flow model represents a possible data flow, i.e., an execution. Which data flow is active depends on the conditions at the control flow elements.

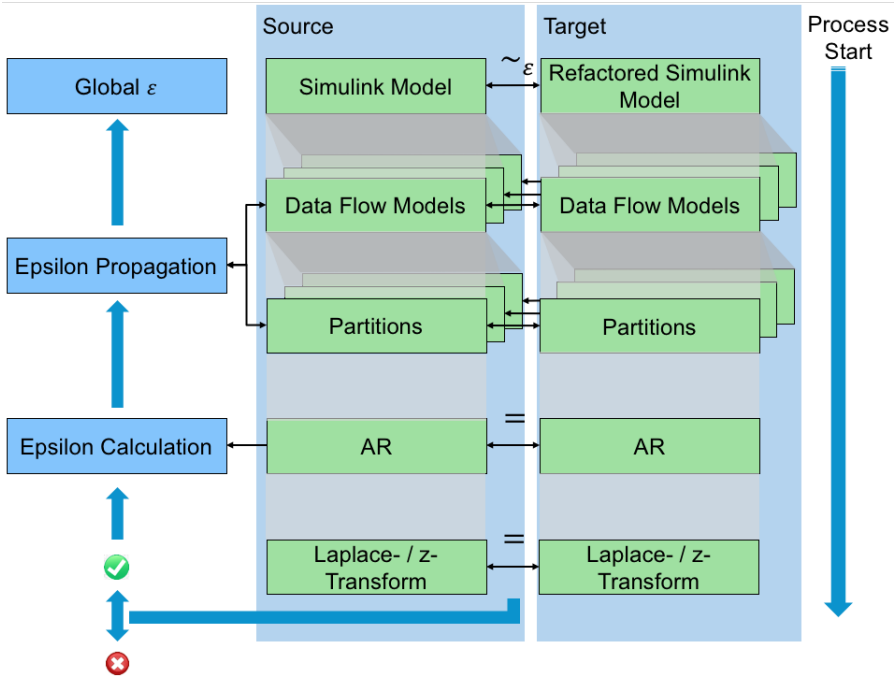

FIGURE 4.2: Overview over the Automation Process

Partitioning In the second step, we partition these data flow models. This means, we split the data flow models into components 1) whose behavioural equivalence can be verified separately and 2) that can be of pairwise different type, time-discrete or time-continuous. We obtain a directed acyclic graph of partitions for each data flow model. These first two steps conclude the syntactical part of our automation approach.

AR Extraction and Symbolic Equivalence Check We extract the ARs of source and target model. We directly calculate the Laplace transform or z-Transform of the ARs. This yields simple algebraic expressions rather than differential or difference equations. Then, we symbolically verify the equivalence of the ARs with the help of a Computer Algebra System (CAS). The symbolic verification follows an *assume-guarantee strategy* within the data flow models along the partitions, which we interpret as directed acyclic graphs.

Epsilon Calculation and Propagation If we are able to establish symbolic equivalence for all data flow models, we calculate the precision of the approximate bisimulation ε. This is performed on AR level to obtain a local ε_l, which is greater than 0 at time-continuous partitions that have been replaced by their analytical solution. This local ε_l is then propagated through the model to 1) obtain a global

ε_g at the points of observation, and 2) to be able to verify the proof obligations at the control flow elements. To obtain an approach for ε propagation, we consider the convergence error of the ODEs and difference equations of partitions subsequent to the partition causing the local ε_l.

4.4 Summary

Our approach consists of two main parts: 1) our theoretical foundation forms a methodology to prove approximate behavioural equivalence of two Simulink models, 2) an automation approach for behavioural equivalence for a subset of Simulink models. The latter is an application of our methodology. Our methodology adapts the notion of approximate bisimulation to Simulink. With this, we adapt an established equivalence notion capable of considering the approximate nature of Simulink models. To enable automated verification, we introduce an Abstract Representation (AR) that describes the behaviour of the models in a denotational manner. We show the soundness of the AR with regard to the operational semantics of Simulink defined in [BC12]. We then provide an approach to derive proof obligations to establish behavioural equivalence. To support an industrially relevant set of semantical refactorings, we extend the operational semantics by various relevant blocks, e.g. the Transfer Function block. We demonstrate an application of our methodology that is semi-automated. Our automation approach uses Laplace and z-transform to enable automated verification. The equality checks and verification of the proof obligations are performed with the help of a Computer Algebra System (CAS).

In the following chapters, we discuss the details of the elements of our approach. We go through each contribution as depicted in Figure 4.1 and the elements in Figure 4.2. We start with the Abstract Representation and the soundness proof with the operational semantics of Simulink in the next chapter. We then continue in Chapter 6 with the adaptation of the approximate bisimulation and the derivation of proof obligations to establish behavioural equivalence of the models. Finally, in Chapter 7, we apply our methodology yielding an automated solution for a subset of Simulink models.

Chapter 5

Abstract Representation - A Denotational Interpretation of Dataflow-Oriented Hybrid Control Systems

The goal of this dissertation is to establish a framework for automated proofs of behavioural equivalence of Simulink models. Moreover, we require our methodology be able to support semantical refactorings such as replacement of submodels by analytical solutions as introduced in Chapter 2. The operational semantics [BC12], which serves as our formal foundation, defines a transition system. Operational semantics evaluate the execution step by step. From a Simulink perspective, this means that the operational semantics describes how the next simulation step, i.e., the values at the upcoming simulation time step, are calculated. However, this format is not suitable for our aim for enabling equivalence proofs for semantical refactorings. One of our objectives is to be able to proof semantical refactorings, such as replacements of ODEs by analytical solutions, as correct. A representation describing an idealised, mathematical behaviour of Simulink's semantics in terms of equations is more suitable to enable automated verification of behavioural equivalence of source and refactored target model. We call this representation Abstract Representation (AR). Solutions of these equations are trajectories. This enables us to compare trajectories of source and target model rather than single steps in the operational semantics to show behavioural equivalence. This in turn enables us to use symbolic approaches via Computer Algebra Systems.

In this chapter, we introduce this AR and prove the soundness with the operational semantics [BC12]. One particular challenge for Simulink, or dataflow-oriented models in general, is that the output function cannot be formulated directly. We can only describe it indirectly with the help of ordinary differential equations (ODEs)

or difference equations. Another challenge is that the operational semantics [BC12] yields behaviour different from solutions of our AR. This is due to the approximate execution or simulation for time-continuous parts. Solutions of our AR are mathematically precise descriptions that describe an idealised behaviour of Simulink models. We show the soundness of our AR and the denotational semantics with respect to the operational semantics for Simulink [BC12] in the following sense: Executions by the operational semantics with decreasing simulation step size yields a sequence of trajectories that uniformly converge against the solutions induced by our AR.

In this chapter, we proceed as follows. Firstly, we describe the AR from syntactical and semantical perspective. Then, we show the soundness with the operational semantics [BC12]. Finally, we extend the operational semantics [BC12] to cope with additional industrially relevant blocks. We do this by reducing the semantics of these blocks to elements we have already considered in the operational semantics and our AR.

5.1 Abstract Representation

In this section, we introduce the Abstract Representation (AR) of a given Simulink model. It is an equation set derived directly from a given Simulink model. Firstly, the equations describe how the output of the blocks are connected with their input, i.e., in what way the respective blocks modify the incoming signals with respect to the simulation time. Secondly, altogether, the equations describe the behaviour of a given model in a denotational style.

In this section, we go through the following items. Firstly, we introduce some syntactical preliminaries. We define Simulink models formally from a syntactical perspective. Then, we define the AR block-wise. The composition of the block-wise equations is the AR of a given model.

5.1.1 Simulink Syntax

We start with a formal definition for Simulink models that emphasises their nature as a graph.

Definition 32 (Simulink Syntax)**.** *A Simulink model is a tuple* $(B, E, I, O, S, C, port, fun, p)$*. B is a finite set of blocks, $E \subset B \times B$ the signal lines, $I \subset B$ the set of input blocks, $O \subset B$ the set of output blocks (for which $\forall b \in O, \forall b' \in B : (b, b') \notin E$ applies), $S \subset B$ the set of stateful blocks, and C the set of control flow blocks. $port : B \times \mathbb{N}^+ \to E \cup \{\bot\}$ is a function that provides the edge $e = port(b, n)$ for the block*

b at the nth position ($\perp$ if the port does not exist).
fun is a function assigning function symbols to blocks in $B\backslash(I \cup O)$, e.g. the sum block is associated with a function symbol $+$.
Moreover, control flow elements $s \in C$ keep a predicate of the form $\diamond\xi$, where $\diamond$ is one of the symbols $<, \leq, =, \geq, >$, and $\xi \in \mathbb{R}$. We denote this predicate with $p(b)$ if $b \in C$.

In this definition, we assume that the model is flattened, i.e., all virtual subsystems have been eliminated. Since virtual subsystems have no impact on the semantics of the Simulink models, this is no restriction. In our implementation, we take care of the prior flattening. Note also that virtual subsystems have input blocks and output blocks themselves, i.e., they form complete Simulink models themselves. Stateful blocks are those with an internal variable during semantical execution. As described in Chapter 2, we distinguish between time-discrete and time-continuous blocks. Examples for the former are Delay block and Discrete Integrator, examples for the latter are Integrator block and Transfer Function block. Examples for control flow blocks are Switch blocks or Multiport Switch blocks. The function $port$ is an auxiliary function that provides the proper edge at the $n-$th place or port. Note that functions associated with the blocks are not commutative in general. Therefore, the order of ingoing signal lines plays an important role. The order is determined by the natural numbers: lower numbers indicate that the signal line is connected before a signal line with a higher number. Elements in I, i.e., input blocks for the model, are only inport blocks in Simulink. This means in particular that we consider other input blocks in Simulink, such as a Sine Wave Generator as arithmetic blocks rather than input blocks. Our elements in I are meant to be interfaces for incoming signals that are known at runtime only.

Next, we define the syntax of expressions. Firstly, we define the language, i.e., the character set of expressions. Then, we define terms and equations. Expressions are used for the AR and to formulate mathematical expressions.

Definition 33 (Language of Expressions)**.** *The language for the Abstract Representation consists of*

- *variable symbols t and h*
- *countably many variable symbols $y_1, y_2, ..., i_1, i_2, l_1, l_2, ..., h_1, h_2, ...$*
- *sequence variable symbols $(y_n)_n, ...$*
- *for every natural number n countably many function symbols $f_1^n, f_2^n, ...$ of arity n.*
- *for every natural number n countably many predicate symbols $p_1^n, p_2^n, ...$ of arity n.*

- *additional symbols* $=, \frac{d}{dt}, ...$

We denote the set of variable symbols except $t, h, h_1, h_2, ...$ *with* $\mathcal{V}$*, i.e.,* $\mathcal{V} = \{y_1, ..., l_1, ..., i_1, ..., a_n, b_n, ...\}$*. Hence, the set* $\mathcal{V}$ *consists of all variable symbols that are associated with a signal line, input variable or stateful block.*

Note that constants are included in this definition as function symbols of arity 0. Furthermore, we use common abbreviations if the context is clear, e.g. the symbol $+$ for the addition instead of a function symbol as in the definition. This also applies to predicates like $\leq$.

We assume that in a given Simulink model, the following variables are assigned to elements of the model as labels:

- Variables of the form l_ν are assigned to each signal line, i.e., each element of E.
- Variables of the form y_ν are assigned to signal lines at the exit of stateful blocks.
- Variables of the form i_ν are assigned to signal lines at the exit of input blocks.

If the context is clear, we identify the variable symbols y_ν, i_ν with the blocks or exiting signal lines interchangeably to keep the notation as simple as possible.

We now define terms and equations inductively.

Definition 34 (Terms and Equations)**.** *Terms are inductively defined as follows.*

- *Variable symbols are terms.*
- *If* $t_1, ..., t_n$ *are terms and* f^n_ν *a function symbol of arity* n *then* $f^n_\nu(t_1, ..., t_n)$ *is a term.*
- *If* $y(t)$ *is a term with variable symbols* y, t*, then* $\frac{d}{dt}y(t)$ *is a term.*

We denote the set of terms with $Term$*. Let* τ *be a term,* y, t, h_ν *variable symbols,* $\lambda \in \mathbb{Z}$*. An equation is of the form*

- $y(t) = \tau$*,*
- $\frac{d}{dt}y(t) = \tau$ *(representing differential equations),*
- $y(t + \lambda h_\nu) = \tau$*, or*
- $a_{n+\lambda} = \tau$ *(representing difference equations by recursive definition of a sequence).*

We denote the set of equations with Eq*.*

The syntax enables us to build the Abstract Representations from Simulink models, which in turn enables us to describe Simulink's behaviour in terms of mathematical expressions.

5.1.2 From Simulink to the Abstract Representation

We describe how we extract the AR from a given Simulink model. The process is twofold. We start with hybrid Simulink models without control flow elements. After that, we extend this definition for Simulink models with control flow elements. We define the AR for hybrid Simulink models without control flow in two stages as well. Firstly, we define the AR per block type. This yields equations describing a functional connection between outgoing and incoming signal variables per block. Secondly, exploiting equalities among these equations in turn allows to compositionally combine the equations to obtain a closed form of our AR. The AR's final closed form for a Simulink model without control flow elements consists of the following items.

- A set of equations to describe the output of the model. The right hand side of the equations depend on the input variables and state variables that occur in the model.
- A set of equations that describe the state of the model. It describes how the state variables relate to themselves and the input variables of the model.

If multiple output blocks or multiple stateful blocks are present in a given model, the AR is denoted in a multi-dimensional vectorial format.

The AR for Simulink models with control elements consists of multiple ARs of Simulink models without control flow elements, which we call data flow models. Each of these data flow models represents a data flow in the given Simulink model under the conjunction of conditions given at its control flow elements.

In the next subsection, we start with the ARs of Simulink models without control flow elements.

AR for Hybrid Simulink Models without Control Flow Elements

We start by defining equations for each block of the model.

Definition 35 (AR for Simulink Models Without Control Flow Elements)**.** *Let $M = (B, E, I, O, S, C, port, fun, p)$ be a Simulink model with $C = \emptyset$. We assume that every element of E is labelled with a variable symbol l_i. To distinguish the input and output edges of the block, we denote the variables with l_{in} and l_{out}. For multiple input signals, we use multi-indices, e.g. l_{in_i}. Let the simulation start be at t_0. We define a function $[\![.]\!]_B : B \to \mathcal{P}(Eq)$ that assigns a set of equations to each block.*

- $b \mapsto \{l_{out}(t) = l_{in}(t)\}$ *if* $b \in I \cup O$.

- $b \mapsto \{l_{out_{n+1}} = l_{in_n}, l_{in_0} = init\}$ *if the delay length of the block is* 1 *(i.e., a Unit Delay block). If the delay length* δ *is greater than* 1*, we introduce auxiliary stateful variables* $a_1, ..., a_{\delta-1}$*. Then*

$$b \mapsto \{l_{out_{n+1}} = a_{\delta-1_n}, ..., a_{1_n} = l_{in_n}, l_{in_0} = init_1, ..., a_{\delta-1_0} = init_\delta\}$$

  $init$ *is the initial value as defined by the user.*

- $b \mapsto \left\{\frac{d}{dt} l_{out}(t) = l_{in}(t), l_{out}(t_0) = init\right\}$ *if* $b \in S$ *is an Integrator block.*

- $b \mapsto \{l_{out}(t) = fun(b)(l_{in_1}(t), l_{in_2}(t), ..., l_{in_n}(t))\}$ *for arithmetic blocks.* $fun(b)$ *is the function symbol associated with block* b*. The order of the variables for incoming signal lines is determined by the auxiliary function* $port$*.*

We define the Abstract Representation (AR) of the model M *to be*

$$[\![M]\!] := \bigcup_{b \in B} [\![b]\!]_B$$

Our AR consists of an equation set for each block of the model. The equations relate the outgoing signal with the incoming signals of the block with respect to the particular function of the block. For time-discrete blocks like the Unit Delay block, we denote sequences, for other blocks we denote functions. We use sequences for time-discrete blocks as a convenient way of describing interval-wise constant functions. Note that the interpretation of a sequence $(a_n)_n$ is a function $\mathbb{N} \to \mathbb{R}$. The natural numbers n indicate the number of the interval $[t_n, t_{n+1}[$ where the value of the time-discrete block remains constant. We discuss this in the section about the interpretation of the AR.

Equalities among equations can be exploited to obtain a closed form of the AR. For instance, if we have equations

$$\left\{l_3(t) = l_1(t) + l_2(t), \frac{d}{dt} l_2(t) = l_3(t), l_2(0) = 1\right\},$$

we can combine this to

$$\left\{\frac{d}{dt} l_2(t) = l_1(t) + l_2(t), l_2(0) = 1\right\}.$$

In particular, for a given hybrid Simulink model without control flow elements, we always obtain the following equations.

1. An equation for the output variables in the form

$$\mathbf{o}(t) = f_o(\mathbf{i}(t), \mathbf{y}(t))$$

Bold variables indicate vectors, i.e., multidimensional elements. The dimension of the vector is equal to the amount of output variables in the model. The variables **i** are input variables in vectorial form, the variables **y** are state variables, i.e., variables at the exit of stateful blocks. We call this equation the **output equation** of the AR.

2. Equations for the stateful blocks or auxiliary stateful variables in the form

$$\mathbf{y}_{n+1} = f_y(\mathbf{i}_n, \mathbf{y}_n), \boldsymbol{y}_0 \text{ the initial value}$$

for the time-discrete case,

$$\frac{d}{dt}\mathbf{y}(t) = f_y(\mathbf{i}(t), \mathbf{y}(t)), \boldsymbol{y}(t_0) = \boldsymbol{y}_0$$

for the time-continuous case.

We call these equations the **state equations** of the AR.

The closed form of the AR is algorithmically extracted from the model by moving backwards in the graph, i.e., along E^{-1} starting at the output blocks. At blocks b, the AR at the block $[\![b]\!]_B$ is calculated. The subexpressions are replaced by this result accordingly. The algorithm stops at 1) input blocks and 2) stateful blocks. All output equations are denoted in vectorial form to obtain the final output equation. To calculate the state equation, the algorithm is repeated starting at stateful blocks and moving backwards from there to build the expressions for the right hand side analogously. We do not introduce a new symbol for the closed form. We use the original AR and the closed form interchangeably if not stated otherwise to keep the notation as simple as possible.

Our AR captures all equations of the Simulink model. The output equations describe the output, i.e., the signals at the output blocks of the model in a denotational form. However, in some cases, we are interested in the trajectory of a signal within the model rather than in the output of the model. This means in particular, we are interested at the AR at the output of a block $b \in B$. In this case, we denote

$$[\![M]\!](b).$$

We mean by this expression all state equations of the model together with the output equation required to describe the output at block b.

In the next subsection, we extend our definition of the AR for hybrid Simulink models without control flow elements to hybrid Simulink models with control flow elements, or simply hybrid Simulink models.

AR for Hybrid Simulink Models

We start with a definition of *configurations* of Simulink models. An intuition behind this is linked with the behaviour of control flow elements. Control flow elements feed through one signal of one of their data ports to their exits. The choice of the signal being fed through depends on the fulfilment of the condition defined at the control flow element. For instance, at a Switch block b, a condition ≥ 0 may be defined. Interpreted by the operational semantics, this yields the following behaviour: If the signal at the control port, i.e., at $port(b, 2)$, fulfills ≥ 0, then the signal at the first data port, i.e., at $port(b, 1)$ is fed through. Otherwise, the signal at the second data port, i.e., at $port(b, 3)$ is fed through. Note that the second data port is actually at the third position of the switch block because the control signal is at the second port. A configuration codifies a data flow in the Simulink model. It consists of a tuple whose elements are either 1 or 2, which represents the data port being fed through at control flow elements.

Definition 36 (Configuration of a Simulink Model)**.** *Let* $M = (B, E, I, O, S, C, port, fun, p)$ *be a hybrid model with control flow,* $C = \{s_1, ..., s_n\}$. *We call an* $n-$ *tuple*

$$\chi := (\chi_1, ..., \chi_n)$$

with

$$\forall 1 \leq i \leq n : \chi_i \in \{1, 2\}$$

a ***configuration*** *of the model. We denote the set of configurations for a model by*

$$X := \{\chi_i | 1 \leq i \leq n\}$$

We now can define data flow models. Data flow models are Simulink models without control flow elements, associated with a configuration. They represent a possible data flow through a given Simulink model. By defining data flow models, we reduce models with control flow elements to a set of models without control flow elements, each representing a possible data flow. A semantical evaluation yields an interval-wise evaluation of the data flow models, i.e., there are intervals in a given evaluation of the semantics where one data flow model is active. The control

signals together with the conditions at control flow elements determines the time steps where changes between active data flow models occur. In our behavioural equivalence verification approach, we compare the data flow models and verify equivalence between data flow models in source and target model with the same configuration. Moreover, we verify the equivalence of the control signals. This altogether yields the equivalence of the overall model. In the following, we define data flow models.

Definition 37 (Data Flow Models)**.** *Let*
$M = (B, E, I, O, S, C, port, fun, p)$ *be a hybrid model with control flow,* $C = \{s_1, ..., s_n\}$, $\chi = (\chi_1, ..., \chi_n)$ *a configuration. A* ***data flow model*** *is a Simulink model*

$$M_\chi := (B_\chi, E_\chi, I, O_\chi, S, \emptyset, port_\chi, fun, \emptyset)$$

without control flow elements with

$$O_\chi := O \cup \{b_{control_i} | 1 \leq i \leq n\}$$

$$E_\chi := (E \backslash \{port(s_i, j) | 1 \leq i \leq n \wedge j \in \{1, 2, 3\}\}) \cup \{(b_{data_{\chi_i}}, b') | (s_i, b') \in E\}$$

where $b_{data_{\chi_i}}$ *is the block* b *with* $(b, s_i) = port(s_i, 1)$ *if* $\chi_i = 1$ *and with* $(b, s_i) = port(s_i, 3)$ *if* $\chi_i = 2$*, and* $b_{control_i}$ *is the block* b *with* $(b, s_i) = port(s_i, 2)$*. Moreover,*

$$B_\chi := I \cup O_\chi \cup S.$$

$port_\chi$ *is accordingly defined to reflect the updated* E_χ*.*

In addition to that, we collect the newly added blocks $b_{control_i}$ *in a set* C_X*, and define a function* p_χ *for* $1 \leq i \leq n$ *with*

$$p_\chi(b_{control_i}) = \diamond b_{control_i} p(s_i)$$

where $\diamond$ *is the negation symbol* $\neg$ *if* $\chi_i = 2$*, and* $\diamond$ *is empty if* $\chi_i = 1$*.*

We define the set of data flow models for a given Simulink model as

$$M_X := \{M_\chi | \chi \in X\}$$

So, the data flow models result from the original model by replacing the control flow blocks by edges from active data ports to subsequent blocks. *Active* means the data port χ_i at control flow block i. At control ports of control flow elements, we add output blocks. The idea is that we also want to observe the signals at these positions because control flow signals together with the conditions defined at the control

flow elements determine the time steps where changes between data flow models occur during semantical evaluation. Moreover, we keep the predicates defined at the control flow elements, and modify them by a negation symbol if $\chi_i = 2$. The reason for this is that in these cases, the configuration of the given data flow model at control flow element i expresses that the second data port is active. Semantically, this is the case if the predicate of the control flow element i is unfulfilled. Note that our approach of modelling control flow elements is strongly connected with the Switch block. In particular, we define the control signal to be at the second port, the data signals at the first and third port, which is the case at Switch blocks. However, other control flow elements follow the same principle. For instance, a Multiport Switch has multiple inports from which the first keeps the control signal. We have aimed for a description that focusses on the conceptual level and keeps the notation simple.

Data flow models do not contain control flow elements anymore. Hence, we can extend our AR we have defined for Simulink models without control flow elements to Simulink models with control flow elements. Our AR for models with control flow elements is the set of ARs of all possible data flow models, which are without control flow elements. To define the AR of data flow models, we can use Definition 35. Together with the ARs of the data flow models, we provide a function that associates the condition under which the given configuration applies. This is the conjunction of the conditions at the control flow elements.

Definition 38 (AR for Simulink Models with Control Flow Elements)**.** *Let $M = (B, E, I, O, S, C, port, fun, p)$ be a hybrid model with control flow, X the set of configurations, M_X the set of data flow models. We define the AR for M as*

$$[\![M]\!] := \left(\{ [\![M_\chi]\!] | \chi \in X \}, \; cond(\chi) = \bigwedge_{i=1}^{n} p_\chi(b_{control_i}) \right) \text{ for } b_{control_i} \in C_X.$$

We provide an example of an AR extraction.

Example 1. *In Figure 5.1, an example is depicted. There, we have only one Switch block. In this case, the configuration is only a 1-tuple. It is either* 1 *or* 2*. The respective data flow models are depicted in Figure 5.2. The AR for the first data flow model is*

$$[\![M_1]\!] = \{out(t) = 2in(t), \frac{d}{dt}y(t) = 2in(t), y(t_0) = 0\}.$$

The AR for the second data flow model is

$$[\![M_2]\!] = \{out(t) = 2y(t), \frac{d}{dt}y(t) = 2y(t), y(t_0) = 0\}.$$

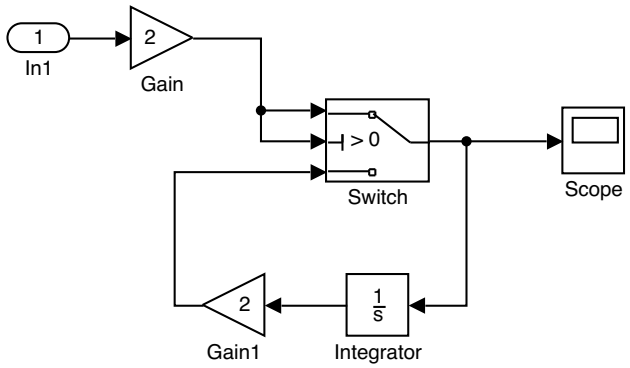

FIGURE 5.1: Example Simulink model with control flow element

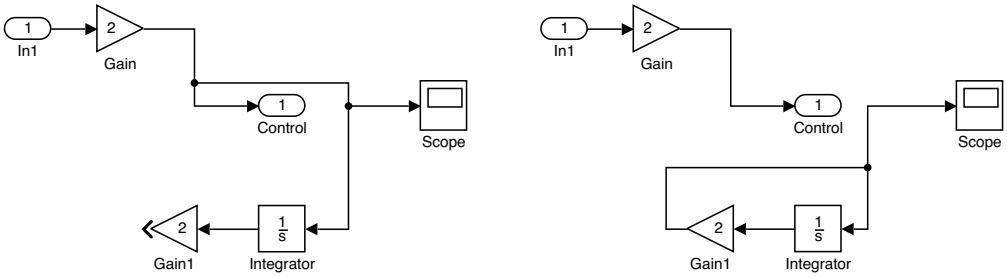

FIGURE 5.2: Data flow models for the example

The conditions for the configurations are

$$cond(1) = 2in(t) > 0, cond(2) = \neg 2in(t) > 0 = 2in(t) \leq 0.$$

We provide a final definition in this subsection to characterise model types by the extracted AR. We use this to distinguish cases in Chapter 6 for showing behavioural conformance of source and target model.

Definition 39 (Simulink Model Types)**.** *Let M be a Simulink model, $[\![M]\!]$ its AR.*

1. *If $[\![M]\!]$ only contains difference equations as state equations, we call the model **purely time-discrete**.*
2. *If $[\![M]\!]$ only contains differential equations as state equations, we call the model **purely time-continuous**.*
3. *If $[\![M]\!]$ contains both types and at least one ODE*

$$\frac{d}{dt}\boldsymbol{y}(t) = f_c(\boldsymbol{y}(t), \boldsymbol{i}(t))$$

with a state variable y_i occurring in the vector $\boldsymbol{y}$ that is described by a difference equation

$$\boldsymbol{y}_{n+1} = f_d(\boldsymbol{y}_n, \boldsymbol{i}_n),$$

*i.e., if ODEs and difference equations share variables, we call the model **time-hybrid**.*

4. *If the model is time-hybrid or contains control flow elements, we call it **hybrid**.*

In the next subsection, we define how the syntactical ARs are interpreted, we define a closed form of the AR, and we define what we understand by an observation of a model.

5.2 Interpretation of the Abstract Representation

In this section, we define how the Abstract Representation is interpreted. Before we come to that, we need to define the notion of *assignment*. An assignment is a function that assigns time-value functions to input blocks.

Definition 40 (Assignment of an Abstract Representation)**.** *Let $M = (B, E, I, O, S, C, port, fun, p)$ be a Simulink model that is being evaluated on the interval $[t_0, t_{end}] =: \mathcal{I} \subset \mathbb{R}$. An **assignment** is a mapping*

$$\varphi : I \to \mathbb{R}^{\mathcal{I}}$$

It assigns a function operating on the simulation interval $\mathcal{I}$ to each input variable.

In other words, the assignment provides the information that is added at runtime at the inport blocks.

In the following, we provide the definition of an interpretation of our AR for hybrid Simulink models. To this end, we define the interpretation for each equation type in the AR. We start with a definition of the interpretation of the AR for Simulink models without control flow elements and extend this to Simulink models with control flow elements. As discussed in Section 4.1, we assume that all time-discrete blocks have the same sample step size h.

Definition 41 (AR Interpretation for hybrid Simulink models without control flow elements)**.** *Let $M = (B, E, I, O, S, \emptyset, port, fun, \emptyset)$ be a hybrid Simulink model without control flow elements that is being executed on the interval $\mathcal{I} \subset \mathbb{R}$, t_0 the simulation start, h the sample step size of the time-discrete elements. Let furthermore $[\![M]\!]$ be the AR and φ an assignment. An **interpretation** of the AR $[\![M]\!]$ under the assignment φ is a function $f : \mathcal{I} \to \mathbb{R}$ that fulfills all equations of $[\![M]\!]$ in the following sense.*

- *Input variables $i(t)$ are being replaced by $\varphi(i)(t)$.*
- *Sequence symbols $(a_n)_n$ are being replaced by interval-wise constant functions $a(t_0 + n \cdot h)$ for $t \in [t_0 + n \cdot h, t_0 + (n+1) \cdot h[$.*
- *Function symbols are replaced by respective functions, e.g. $+$ is replaced by the function that adds real numbers.*

If an $f : \mathcal{I} \to \mathbb{R}$ exists that fulfills all of the AR's equations, i.e., that solves the difference and differential equations with initital value y_0 at t_0, we denote this by

$$f \models [\![M]\!]^{\varphi;t_0,y_0}.$$

If the initial value is clear from the context, we can also briefly denote

$$f \models [\![M]\!]^{\varphi}.$$

f is an interval-wise defined function. The intervals are $[t_n, t_{n+1}[$ with $t_n = t_0 + n \cdot h$. If M is purely continuous, it is a continuous function on the whole interval $\mathcal{I}$.

If the model is purely time-discrete, the AR consists only of time-discrete blocks yielding a set of difference equations and the output equation. In this case, the interpretation is a sequence solving all difference equations, which in turn are interpreted by interval-wise constant functions. If the model is purely time-continuous, the AR consists of ODEs and the output equations. In this case, the interpretation is a continuous function on $\mathcal{I}$ fulfilling all ODEs and the output equations. If the model consists of both time-discrete, and time-continuous blocks, the AR consists of both types difference equations and ODEs. In this case, the interpretation is an interval-wise function. In each interval $[t_n, t_{n+1}[$, the respective ODE $\frac{d}{dt}\boldsymbol{y}(t) = f_y(\boldsymbol{y}(t), \boldsymbol{i}(t))$ is fulfilled where some of the elements in the vector $\boldsymbol{y}(t)$ are constants determined by the solution of the difference equations from the time-discrete elements.

We now extend this definition of the interpretation of our AR for Simulink models without control flow to Simulink models with control flow.

Definition 42 (AR Interpretation for hybrid Simulink models)**.** *Let $M = (B, E, I, O, S, C, port, fun, p)$ be a hybrid Simulink model that is being executed on the interval $\mathcal{I} \subset \mathbb{R}$, t_0 the simulation start, h the sample step size of the time-discrete elements. Let furthermore $[\![M]\!]$ be the AR and φ an assignment.*

The interpretation of the AR $[\![M]\!]^{\varphi}$ of the model M is defined as an interval-wise defined function $f(t) := f_i(t)$ if $t \in \mathcal{I}_i$ where $\mathcal{I}_i$ are intervals partitioning $\mathcal{I}$, i.e., $\mathcal{I}_i = [t_{s_i}, t_{s_{i+1}}[$

with $t_{s_0} = t_0$ (t_s for switching time) such that for each function $f_i : \mathcal{I}_i \to \mathbb{R}$ holds

$$f_i \models [\![M_\chi]\!]^{\varphi;t_{s_i},f_{y_i}(f_{i-1}(t_{s_i}),\varphi(i)(t_{s_i}))}, f_0 \models [\![M]\!]^{\varphi,t_0,y_0}_\chi$$

for a configuration χ with

$$\models cond(\chi)[b_{control_j} \leftarrow f_{control_{j_i}}] \text{ for } f_{control_{j_i}} \models [\![M_\chi]\!]^{\varphi;t_{s_i},f_{y_i}(f_{i-1}(t_{s_i}),\varphi(i)(t_{s_i}))}(b_{control_j}).$$

Here, $\models cond(\chi)$ means that the condition is fulfilled in the sense of predicate logic, and $cond(\chi)[b_{control_j} \leftarrow f_{control_j}]$ means that in $cond(\chi)$, $b_{control_j}$ gets replaced by $f_{control_j}$. The function f_{y_i} determining the initial value for the ODE or difference equation of the AR in the next data flow model is the respective state equation in this data flow model.

An interpretation of the AR of a hybrid Simulink model with control flow is reduced to the interpretation of the data flow models. Which data flow model applies is determined by the condition defined at control flow elements. It is the conjunction of the predicates that the interpretations of the control signals fulfill.

When a switch occurs, the change from configuration χ_1 to χ_2 has two implications: Firstly, the equations determined by the AR change - the trajectory in the subsequent interval follows then $[\![M_{\chi_2}]\!]$. Secondly, the initial value for the state equations, i.e., the ODEs and difference equations, is determined by the value of the trajectory of the preceding configuration, at the switching time step t_s. We have denoted this by stating

$$f_i \models [\![M_\chi]\!]^{\varphi;t_{s_i},f_{y_i}(f_{i-1}(t_{s_i}),\varphi(i)(t_{s_i}))}.$$

Note that unlike in control flow-oriented languages, in data flow-oriented languages, the submodel at an inactive port is evaluated with evolving simulation time. Particularly, this is of importance if the AR of two data flow models share state variables.

We illustrate this behaviour with an informal example.

Example 2. *Figure 5.3 depicts an example of a hybrid Simulink model with a Switch block. At the beginning of the execution (at $t_0 = 0$), data port 2 is fed through because the condition is not fulfilled, i.e., the second data flow model is evaluated. The output is $t + 1$ because the Ramp block returns the values t at execution time t. From $t = 2$, the condition is fulfilled and therefore the values from data port 1 are fed through, i.e., data flow model 1 is evaluated. The behaviour of the system at data port 1 can be described by the ODE $\frac{d}{dt}y(t) = y(t)$. However, during the evaluation for $t \in [0, 2[$, the path from the output of the Switch block to the Integrator block and finally to data port 1 is also executed. This is consistent with the*

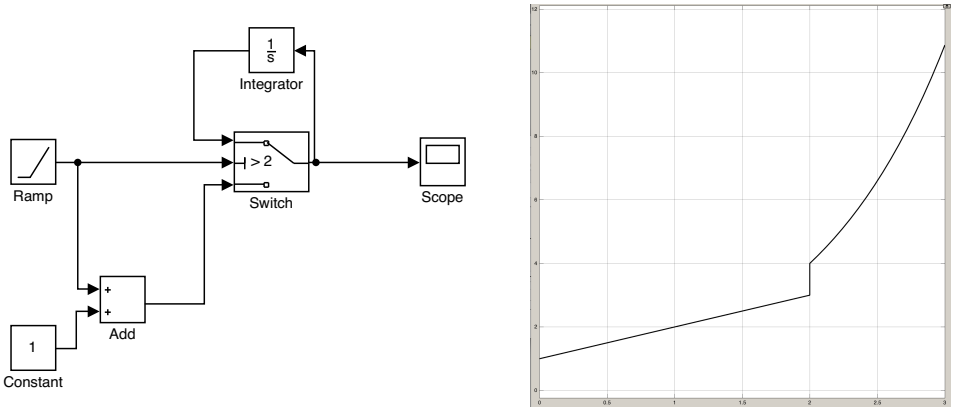

FIGURE 5.3: Hybrid Example With Control Flow and Simulation Output

operational semantics. However, it may be counterintuitive because we may expect that the whole data path at data port 1 is not executed. The operational semantics however states that all blocks are still executed. Just the data from data port 1 to the the output of the Switch block is blocked. This means that for $t \in [0, 2[$*, the value at the exit of the Integrator block is*

$$y(t) = \int_0^t (\tau + 1) d\tau$$

expressed as formula in precise mathematical way as we have introduced by the AR. Hence, the intial value at time step 2 *is*

$$\int_0^2 (t + 1) dt = 4$$

and the solution of the ODE is given as

$$y(t) = out(t) = 4e^{t-2}$$

This is consistent with the output at the right hand side of Figure 5.3. The graph jumps to 4 *at* $t = 2$ *and the curve follows* $y(t) = 4e^{t-2}$*.*

Note that the value of the preceding trajectory at switching time t_s is actually only relevant in the time-hybrid case. If data flow models do not share state variables, the initial value at t_s just depends on the values of the input signals at t_s.

Our definition of the AR and the interpretation respects this behaviour. This argumentation also provides a justification for our decision to always extract state equations in all data flow models, also if they may not impact directly the output of a data flow model. Moreover, as we discuss in the next Chapter, our AR contains sufficient information that we use to conclude behavioural conformance of source

and refactored target model.

In the next section, we show the soundness of our AR interpretation with respect to the operational semantics [BC12].

5.3 Soundness of Abstract Representations with Operational Semantics

In this section, we clarify the relation between the AR and the transition system from the operational semantics [BC12]. We show the soundness for time-discrete and time-continuous block types. Then, we extend this to the soundness for models without control flow. Finally, we discuss soundness for hybrid models.

To improve readability and avoid distractions of the reader, we refer the proofs to Appendix A. We provide the idea of the proofs directly in this section.

To establish a comparability between the values from an evaluation of the operational semantics for a Simulink model as described in [BC12], and interpretations of our AR, we introduce the notion of trajectories of a semantical evaluation.

As described in Chapter 2, semantical states in Simulink are mappings $\sigma : V \to \mathbb{R}$ where $V = \{l_i\} \cup \{d_i\} \cup \{x_i\} \cup \{t, h\}$. The l_i, d_i, x_i are variables associated with the exit of the blocks and internal variables for stateful blocks. The distinguished variables t and h stand for the simulation time and global simulation step size. Additionally, every block can have parameters associated, e.g. the sample step size of a time-discrete block. In essence, a semantical state is a snapshot that carries the information of the current simulation step. The operational semantics [BC12] defines how the states change as the simulation time increases by providing inference rules. Hence, an evaluation of the operational semantics yields a function $\mathcal{I} \to \mathbb{R}$ at each signal line. We call it trajectory at the signal line. The symbol $\mathcal{I}$ is the simulation interval. The *trajectory* of a signal line l_i is the result of the complete semantical evaluation.

Definition 43 (Trajectory of Semantical Evaluation of Simulink Models)**.** *Let $M = (B, E, I, O, S, C, port)$ be a Simulink model, interpreted by a transition system defined by the operational semantics [BC12] in the interval $\mathcal{I}$. Let Σ_M be the set of states of this execution for model M. For a block $b \in B$ with associated output variable v, we denote*

$$[\![.]\!]_{OS}(b)(\tau) \in \mathbb{R}^{\mathcal{I}}, M \mapsto \sigma(v) \text{ where } \sigma \in \Sigma_M \text{ such that } \sigma(t) = \tau$$

OS *stands for operational semantics in the formula. Note that in this formula, it is implicitly assumed that for all $b \in I$, we have values $\sigma(v)$ available for all $\tau \in \mathcal{I}$ as well. In other words, we implicitly assume that the model has been instantiated with signals at the input*

blocks. An assignment φ determines the values of $[\![M]\!]_{OS}(b)(\tau)$ for all $\tau \in \mathcal{I}$ and $b \in I$. We denote this circumstance with $[\![M]\!]^{\varphi}_{OS}(b)(\tau)$ and call it the execution of the model M at block $b \in B$ under the assignment φ.

The graph

$$t_b = \{(\tau, y) | \tau \in \mathcal{I} \wedge [\![M]\!]_{OS}(b)(\tau) = y\}$$

*is called **trajectory** of b.*

In essence, the trajectory describes the graph at the exit of b and is determined by the sequence of all values for the associated variable of b at all time steps.

To prove soundness of our AR with respect to the operational semantics, we show that the trajectory determined by $[\![M]\!]^{\varphi}_{OS}(b)$ for a block b fulfills the equations of our AR at block b. Since we have a synchronous semantics, this in turn yields the soundness for the overall model.

We proceed by showing the soundness of our AR for time-discrete blocks. This means, we show that the relation between outgoing and incoming signals of a block as defined by the respective equation in the AR is fulfilled. We do not show that the interpretation of the AR is unique at this point. We require the uniqueness in the next chapter to prove behavioural equivalence.

Lemma 16 (Soundness of AR for Time-discrete Blocks)**.** *Let $M = (B, E, I, O, S, \emptyset, port, fun, \emptyset)$ be a time-discrete Simulink model that is being evaluated in the interval $\mathcal{I}$ following the operational semantics [BC12], time-discrete blocks with sample step size h, simulation start at t_0. Let furthermore be $b \in B$, $[\![M]\!]^{\varphi}$ the AR under an assignment φ, and $out(t) = f(\boldsymbol{i}(t))$ be the equation associated with block b. Then the following holds.*

$$\forall t \in \mathcal{I} : [\![M]\!]^{\varphi}_{OS}(b)(t) = f(\varphi(\boldsymbol{i})(t))$$

The proof is performed by performing a case distinction by block types. For input blocks, output blocks, and arithmetic blocks, only major step transitions apply. Please refer to Chapter 2 for an overview over the transitions of the operational semantics. They are evaluated at sample steps h and at these sample times, they directly feed through the ingoing signal. Input blocks feed through the signal determined by the assignment φ at sample steps. In case of arithmetic blocks, the arithmetic function is evaluated, the result fed through to the output. Between sample steps, the output of the blocks remain constant. For time-discrete blocks like the delay block, the ingoing signal is internally stored in an internal variable. This happens at sample time steps as well. The output is determined by the value of the internal variable. Since according to the global simulation rule, the calculation of the outputs (major step transition) is applied before the update of the internal variables,

this results in a delay, which is expressed by the difference equations. A detailed proof can be found in Appendix A.

We proceed with the soundness for time-continuous blocks. The challenge for the time-continuous case is that we cannot show that the equations of the AR are fulfilled by the trajectory of the semantical execution directly. The semantical execution is always performed with a finite sample step size, i.e., we always obtain a discretized solution. Notions that occur in the AR like $\frac{d}{dt}y$ do not make sense if naively applied to the trajectory of the semantical execution. We construct a limit function for which a derivative makes sense and the AR's equation is fulfilled. In the following lemma, we restrict time-continuous blocks to Integrator blocks. The reason for that is that Integrator blocks conceptually exhibit the relevant attributes of time-continuous blocks. In the subsequent section, we additionally consider more complex time-continuous blocks like Transfer Function blocks. We extend the operational semantics [BC12] to support these blocks. To this end, we reduce them to subsystems that only consist of arithmetic blocks and Integrator blocks. We formulate the following lemma for the case where the Euler technique has been chosen as approximation method to keep the notation simple. For other, consistent approximation techniques, e.g. Runge Kutta techniques, the statement holds and the proof works analogously. Please refer to Section 2.2.5 for an overview over numerical approximation.

Lemma 17 (Soundness of AR for Time-continuous Blocks)**.** *Let $M = (B, E, I, O, S, \emptyset, port, fun, \emptyset)$ be a time-continuous Simulink model. that is being executed in the interval $\mathcal{I}$ following the operational semantics [BC12], simluation start at t_0 under the assignment φ. We assume that $Im(\varphi) \subset C(\mathcal{I}, \mathbb{R})$, i.e., the input blocks are instantiated with continuous signals. Moreover, $Im(\varphi)$ are assumed to be piece-wise bounded functions.*

Let furthermore be $b \in S$ an Integrator block, $[\![M]\!]^{\varphi}$ the AR under an assignment φ.

For a sample step size h, we associate a function $[\![M]\!]^{\varphi,h}_{OS}(b)$ as follows.

$$[\![M]\!]^{\varphi,h}_{OS}(b)(\tau) = \frac{\sigma_{n+1}(v) - \sigma_n(v)}{h}(\tau - t_n) + \sigma_n(v) \text{ for } \tau \in [t_n, t_{n+1}[$$

where $\sigma_n, \sigma_{n+1} \in \Sigma_M$ such that $\sigma_n(t) = t_n$, $\sigma_{n+1}(t) = t_{n+1}$, $t_{n+1} = t_n + h$, and v being the variable associated with block b. This function yields the same values as the execution by the operational semantics $[\![M]\!]_{OS}(b)$ for block b at time steps $t_n = t_0 + n \cdot h$ if the Euler method has been chosen as approximation method by the user in Simulink.

Between time steps, i.e., for $t \in]t_n, t_{n+1}[$, the function follows a linear curve connecting the values at time steps t_n, t_{n+1} rather than being constant as in the trajectory of $[\![M]\!]_{OS}(b)$. We assume that the trajectory of the function $[\![M]\!]^{\varphi,h}_{OS}(b)$ is bounded on $\mathcal{I}$.

Every sequence in the family

$$\mathcal{F} := \left\{ [\![M]\!]_{OS}^{\varphi,h}(b) \middle| h = \frac{\#\mathcal{I}}{n} \text{ for } n \in \mathbb{N}^+ \right\}$$

of evaluations with fixed sample step sizes contains a subsequence that converges uniformly to a function f if $\lim_{n\to\infty} h_n = 0$. For this function the following holds.

$$f \models [\![M]\!]^{\varphi;t_0,y_0}(b)$$

We also denote this relationship between $[\![M]\!]_{OS}^{\varphi}(b)$ and $[\![M]\!]^{\varphi}(b)$ with

$$[\![M]\!]_{OS}^{\varphi}(b) \models_{h\to 0} [\![M]\!]^{\varphi}(b)$$

For the detailed proof, we refer to Appendix A. The proof idea is as follows. The function $[\![M]\!]_{OS}^{\varphi,h}(b)$ constructs an Euler polygonial curve. For a monotonically decreasing sequence of sample step sizes $(h_k)_k$, we obtain a respective function $[\![M]\!]_{OS}^{\varphi,h_k}(b)$. This forms the family or set of functions $\mathcal{F}$. Analogously to the Theorem of Peano [Aul04], we show that this family of functions is equicontinuous, and piecewise-bounded. This allows us to apply the Theorem of Arzelà-Ascoli [Tim08]. It says that there exists a limit function to which the family of functions $\mathcal{F}$ converges uniformly. For this limit function, we show then that it solves the ODE of the Integrator block and therefore fulfills the equation of the AR.

Note that the boundedness of the input signal of the Integrator block b can be guaranteed if the input signals of the model, i.e., $\varphi(i)$ is bounded for all input blocks i, and the function $f(\boldsymbol{y}(t), \boldsymbol{i}(t))$ describing the input of block b is continuous.

In Lemma 17, we have shown the soundness of the AR for time-continuous blocks if the Euler method has been chosen as approximation technique in Simulink. The proof works analogously for other approximation techniques. The difference is that in order to solve the ODE

$$\frac{d}{dt}y(t) = f(t, y(t)), y(t_0) = y_0$$

the sequence for the approximation follows

$$y_{n+1} = \Phi(t_n, y_n, h)$$

For the Euler method,

$$\Phi(t_n, y_n, h) = hf(t_n, y_n)$$

The technique is consistent if the approximation converges to the solution as $h \to 0$.

The approximation techniques used in Simulink are all consistent. In summary, this means that for an approximation technique different from Euler, our construction of $[\![M]\!]^{\varphi,h}_{OS}(b)$ needs to be amended according to the chosen technique. However, the argumentation of the proof would remain the same.

The soundness for time-discrete and time-continuous blocks as shown in Lemmas 16 and 17, allows us to conclude the soundness for hybrid Simulink models as follows: For a given model M, a block b and the set of configurations X, we can for every $\chi \in X$ construct a limit function for the evaluation of the operational semantics [BC12] for every sequence of decreasing sample step sizes. This function f_χ fulfills both the ODEs and difference equations of the AR $[\![M_\chi]\!]^\varphi$ for an arbitrary but fixed assignment φ that is piecewise continuous. This function f_χ follows a time-continuous trajectory for time-continuous stateful blocks b (or arithmetic blocks b depending on time-continuous blocks) in intervals $]t_n, t_{n+1}[$. The time-discrete steps t_n are assumed to be fixed and equal among all time-discrete blocks. At time steps t_n, a discrete jump occurs due to updates from time-discrete blocks. The output at time-discrete blocks is constant in the intervals. Since due to the operational semantics, all major step transitions, all update transitions, and all solver transitions, are applied in parallel, i.e., we have a synchronous semantics, we can conclude that

$$f_\chi \models [\![M_\chi]\!]^\varphi(b).$$

This means, for every configuration, the AR describes the idealised behaviour, i.e., the output of the constructed limiting function f_χ correctly.

The configuration that applies is determined by the conditions at the control flow elements. The relevant semantical rules that apply here is

$$\frac{\langle e_1, \sigma\rangle \xrightarrow{o} true\, \langle e_2, \sigma\rangle \xrightarrow{o} r_2}{\langle if(e_1, e_2, e_3), \sigma\rangle \xrightarrow{o} r_2} \quad \frac{\langle e_1, \sigma\rangle \xrightarrow{o} false\, \langle e_3, \sigma\rangle \xrightarrow{o} r_3}{\langle if(e_1, e_2, e_3), \sigma\rangle \xrightarrow{o} r_3}$$

as described in Chapter 2. The conditional statements are extracted at control flow elements as described in the operational semantics [BC12]. In these rules, the control signal has been associated with the variable e_1, the data flow signals have been associated with e_2, e_3. In our AR, we have the control signal associated with the second port, the first data signal with the first port, and the second data signal with the third data port, which is consistent with the Switch block in Simulink. In the configuration χ, the $i-th$ position determines the true or false case at the $i-th$ control flow element. Hence, the conjunction of all conditions, modified by $\neg$ if $\chi_i = 2$, which expresses the negative case, describes correctly the condition for which this particular data flow model applies. Moreover, the construction of

our data flow models as in Definition 37 reproduces correctly the semantics of the inference rule for control flow elements as given above. The transfer of initial values between configurations is also considered in our AR as described in the supplement of Definition 42.

We can therefore conclude the soundness of our AR with the operational semantics for hybrid Simulink models.

In the next section, we extend our AR to cope with more complex Simulink elements, which are of high relevance for industrially motivated models.

5.4 Extension of the Operational Semantics for Simulink

To be able to support a relevant subset of Simulink models, we extend the operational semantics [BC12] by various blocks. By extension we mean that we enable the operational semantics to cope with more complex blocks. We do not mean to introduce new inference rules. Instead, we reduce the blocks we add to subsystems that only consist of blocks we already support and declare the extension as the behaviour of these subsystems. We also reason why these extensions, or alternative expressions by these subsystems are sound.

We start with (Discrete) Transfer Function blocks, continue with stateful blocks, and close with some arithmetic and logical blocks. The key idea is to reduce the additional blocks to elements that have been already considered in the operational semantics [BC12] and consequently also in our Abstract Representation.

Extension by (Discrete) Transfer Function Blocks

In this section, we extend our formal foundation and explain the semantics of Transfer Function blocks. Transfer Function blocks are time-continuous stateful blocks. They are linked with the Laplace transform. The time-discrete analogon are the Discrete Transfer Function blocks. They are linked with the z-transform. The concept for Discrete Transfer Function blocks is analogous to the concept for Transfer Function blocks. We therefore restrict our argumentation to Transfer Function blocks.

Syntactically, Transfer Function blocks have one inport only and show a rational polynomial $\frac{p(s)}{q(s)}$.

Regarding semantics, the intuitive idea is that initially the incoming signal l_{in} is transformed via Laplace transform or z-transform yielding a function in the complex domain $\mathcal{L}(l_{in})$. Then, the result is multiplied with the polynomial $\frac{p(s)}{q(s)}$ yielding $\frac{p(s)}{q(s)} \cdot \mathcal{L}(l_{in})$. Finally this is inverted to obtain a signal at the output of the Transfer Function block in time domain, i.e., $\mathcal{L}^{-1}(\mathcal{L}(l_{in}))$. The idea behind this comes

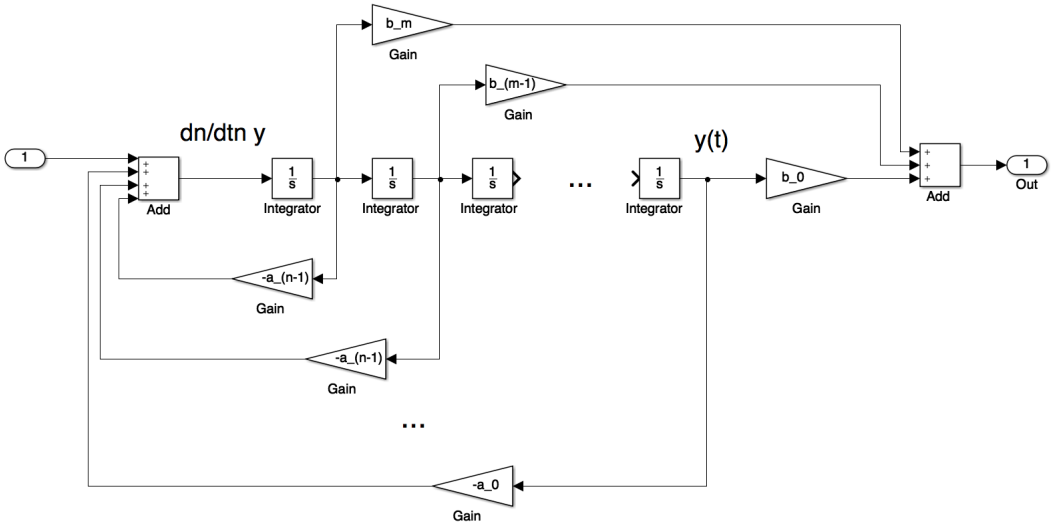

FIGURE 5.4: Equivalent submodel for Transfer Function Block

from control theory [Kai80]. By introducing the transfer function, control theory provides a direct link between output and input signals in the complex-valued *frequency* domain. Self-references as in ODEs, which in turn are feedback loops in Simulink models, are hidden. This in turn is a strong motivation for refactorings, which often aim for complexity reduction of the models.

Our extension is twofold. Firstly, we replace a Transfer Function block by a mathematically equivalent ODE system with constant coefficients. Secondly, we express this ODE system as Simulink model and explain the semantics of the Transfer Function block as the semantics of this model. In essence, we reduce the semantics of the Transfer Function block to a model with elements that have already been defined in the operational semantics.

For a Transfer Function block with the inscription

$$\frac{b_m s^m + ... + b_1 s + b_0}{a_n s^n + a_{n-1} s^{n-1} + ... + a_1 s + a_0}$$

we associate the submodel depicted in Figure 5.4. All initial parameters of the Integrators are set to 0. The inport and outport blocks of the depicted submode coincide with the incoming and outgoing ports of the transfer function block. We declare the semantics of the transfer function block by applying the rules from the operational semantics [BC12] on the associated submodel.

In the following, we justify that the submodel given in the definition is mathematically equivalent to the Transfer Function block.

To this end, we associate to a Transfer Function block with inscription

$$\frac{b_m s^m + ... + b_1 s + b_0}{a_n s^n + a_{n-1} s^{n-1} + ... + a_1 s + a_0}$$

the following ODE:

$$\frac{d^n}{dt^n} y = -\sum_{\nu=0}^{n-1} a_\nu \frac{d^\nu}{dt^\nu} y + \sum_{\mu=0}^{m-1} \frac{d^\mu}{dt^\mu} in(t) \tag{5.1}$$

with $y(0) = \frac{d}{dt}y(0) = ... = \frac{d^n}{dt^n}y(0) = 0$, where y denotes the outgoing signal and in the incoming signal. Then, for continuous $in : \mathcal{I} \to \mathbb{C}$ with exponential growth of rate at most α, the following holds:

$$\exists \beta > 0 : \forall s \in \mathbb{C}, Re(s) > \beta : \mathcal{L}(y)(s) = \frac{b_m s^m + ... + b_1 s + b_0}{a_n s^n + a_{n-1} s^{n-1} + ... + a_1 s + a_0} \mathcal{L}(in)(s)$$

if and only if $y : \mathcal{I} \to \mathbb{C}$ is the unique solution of the ODE as above. The proof for this claim follows directly by applying Laplace transform on both sides of equation 5.4. Since the solution of the ODE is unique and the Laplace transform is injective in this case, the equivalence of ODE and Transfer Function follows directly. Moreover, Figure 5.4 is a direct expression of the ODE system.

Extension by Additional Stateful Blocks

In this subsection, we provide some extensions for additional stateful blocks. It follows the same principle as for Transfer Function blocks: We provide alternative expressions for the blocks by using only arithmetic blocks, Switch blocks, and already covered stateful blocks, i.e., the Unit Delay and Integrator block. Then, we declare their semantics to be the semantics of the equivalent system executed by the operational semantics [BC12]. We describe alternative expressions for two representatives. The principle of construction that we demonstrate gives an indication how this can be used to construct other stateful blocks if required.

We start with the Discrete-Time Integrator. It is a representative for time-discrete blocks. It has therefore a sample step size h. Between time steps, the value is kept constant, at time steps $t_n = t_0 + h$, it changes its value at its exit. The block has two parameters: a gain, i.e., a real value, and an initial value. Let the initial value be 0. The semantics of the block is as follows: At sample step times, it feeds through the internally stored value multiplied by the gain value. Moreover, it adds the internally stored value to the value of the ingoing signal, and stores this result internally. Hence, we can formulate the equations in the style of the syntactical equations used

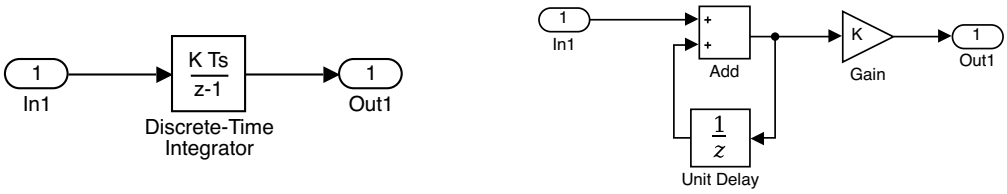

FIGURE 5.5: Discrete-Time Integator Block and equivalent expression

FIGURE 5.6: Delay Block and equivalent expression

in the operational semantics [BC12] (cf. Section 2.3.2) as follows:

$$l_{out} := \lambda \cdot l_1; l_1 :=_S d; d :=_S d + l_{in}, d_{init} = 0$$

In the formula, we denote λ as the gain value. This results in AR equations

$$l_{out_{n+1}} = \lambda l_{1_n}, l_{1_{n+1}} = l_{in_n} + l_{1_n}, l_{1_0} = 0$$

Figure 5.5 shows the equivalent models.

Another example is a Delay block with delay > 1. We equivalently express a Delay block with delay δ by introducing δ Unit Delay blocks.

Figure 5.6 depicts the equivalent systems.

Extension by Additional Direct-Feedthrough Blocks

In this section, we proceed with our extension and add direct-feedthrough blocks.

Logical Blocks We start with logical blocks, such as Not, And, and Or. We express them equivalently by using only arithmetic blocks and Switch blocks. All blocks in this paragraph are direct-feedthrough blocks.

The Not block has one inport. The semantics is as follows: If the ingoing signal is equal to 0, the output is 1. Otherwise, the output is 0. Figure 5.7 depicts the Not block and an alternative expression. The inport blocks are to carry the same signal. The condition at the Switch block is $\neq 0$, i.e., it checks if the signal is not equal to 0. If this is true, then the constant 0 is fed through, otherwise the constant 1. This expression fulfills obviously the semantics of the Not block.

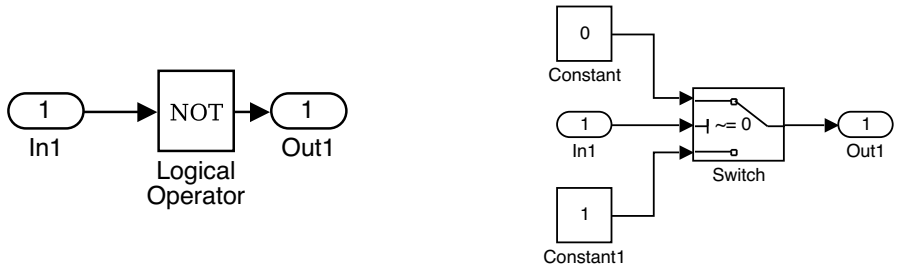

FIGURE 5.7: Equivalent expressions: Not block

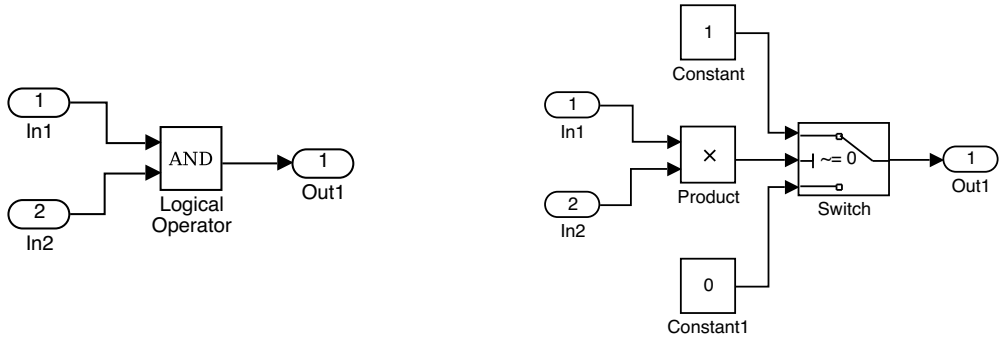

FIGURE 5.8: Equivalent expressions: And block

Figure 5.8 provides an alternative expression for the And block. Its semantics is as follows: Its output is 1 if both incoming signals are not 0, otherwise the output is 0. The product joins the signals and becomes 0 if one of the signals are 0. If this is the case, 0 is fed through the Switch block, otherwise, i.e., if both are not 0, the output of the Switch block is 1.

Finally, we consider the Or block. Its semantics is that it is 1 if one of the incoming signals is $\neq 0$. Otherwise it is 0. Since $a \vee b = \neg(\neg a \wedge \neg b)$, we can express this by the constructs we already have defined.

Arithmetic Blocks We want to demonstrate our extension by two additional direct-feedthrough block, the Saturation block, and the absolute value block. The Saturation block has two parameters: an upper and a lower threshold. It directly feeds through the values of the ingoing signal if they are between the two thresholds. However, if the incoming value exceeds the upper or falls under the lower limit, the values are kept at these limits.

Figure 5.9 shows the two equivalent models. There, ut denotes the upper threshold, lt denotes the lower threshold.

The absolute value block calculates the absolute value of the incoming signal. This means particularly that if the value of the incoming signal is ≥ 0, then the

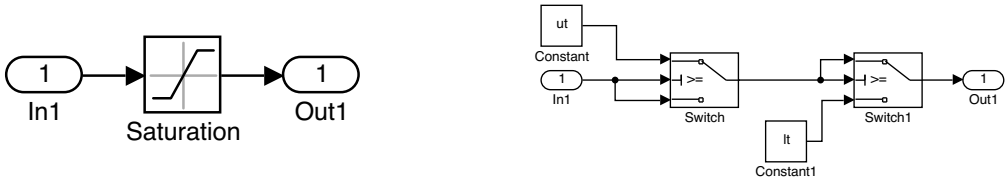

FIGURE 5.9: Saturation Block and equivalent expression

FIGURE 5.10: Abs Block and equivalent expression

signal is directly fed through, otherwise the signal multiplied with -1. Figure 5.10 is a direct expression of this semantics.

This concludes our extension of the operational semantics. The examples are not a comprehensive list, but they demonstrate the principle how our approach can be used to cope with a broad variety of Simulink blocks. We are confident that this list could be extended to be embedded in our approach. We leave this to future work.

5.5 Summary

In this chapter, we have described the extraction of a system of equations for Simulink models, which we call the *Abstract Representation* (AR). This system of equations describes the behaviour of Simulink models in a denotational style, i.e., the trajectory resulting from the execution of the operational semantics on the whole simulation interval fulfills the equations of the AR. The AR consists of sequences for time-discrete parts and continuously evolving functions for time-continuous parts. Sequences are specified by difference equations, functions are specified by ordinary differential equations. We have proven the soundness of this representation with respect to the operational semantics as described in [BC12]. For time-discrete models, the trajectory of the model fulfills the equations of the AR directly. For time-continuous and hybrid models, we have constructed a sequence of trajectories with decreasing simulation step size that converges uniformly to a function that fulfills the equations of the AR.

The denotational style of the AR enables automated verification of correctness of refactorings on Simulink models. In particular, it enables us to compare trajectories

of solutions of our AR for verification of behavioural equivalence rather than comparing single steps in the operational semantics. This in turn, enables us to identify semantic refactorings as being correct such as a replacement of a model representing an ODE by a model representing the analytical solution of the ODE.

To cope with more complex, industrially relevant blocks, we have extended our AR. The key idea is to equivalently express blocks by elements we have already considered in the operational semantics and our AR.

In the next chapter, we describe how we use the AR to verify behavioural conformance of source and refactored target model.

Chapter 6

Approximate Bisimulation for Simulink Models

In this chapter, we present our adaption of the concept of approximate bisimulation to Simulink. Approximate bisimulation is an established equivalence notion capable of expressing differences between values that are evaluated by a semantics. Two states are equivalent with respect to the approximate bisimulation of precision ε if their values differ by at most ε and their next states according to the operational semantics are again equivalent. We use this equivalence notion to be able to consider the approxiate nature of the semantics of data flow-oriented or signal flow-oriented languages for hybrid control models, such as Simulink.

After our adaptation of approximate bisimulation to Simulink, we provide sufficient conditions for approximate bisimulation based on our AR. With our AR, we are able to reason on a mathematical level. This allows us to identify semantical refactorings such as analytical solutions of ODEs as correct. However, due to the approximate nature of Simulink's semantics, time-continuous parts are approximated as we have seen in Chapter 5. The key idea to conclude approximate bisimulation is: Firstly, to symbolically verify equivalence with the AR. Secondly, to calculate an estimate of the approximation, i.e., the difference between evaluated values in Simulink and the mathematical solution of our AR. This in turn provides an estimate for the precision ε of the approximate bisimulation. For cases $\varepsilon > 0$, the estimate ε increases with the simulation interval length $\#\mathcal{I}$, which we denote by $\varepsilon(\#\mathcal{I})$ if we want to emphasise this dependency.

6.1 Adaptation of Approximate Bisimulation for Simulink

In this section, we adapt the concept of approximate bisimulation to Simulink. Approximate bisimulation weakens the definition of traditional bisimulation. While traditional bisimulation demands the evaluated values of equivalent states after one

step to be equal, approximate bisimulation allows them to be close to each other up to a maximal distance ε. This ε is called precision of the approximate bisimulation. For $\varepsilon = 0$, we have traditional bisimulation. Approximate bisimulation has been presented e.g. in [GP11, GP07b, Gir05] and is an established concept in the hybrid systems community.

In the following, we start by introducing the concept of approximate bisimulation. Then, we adapt it to the operational semantics of Simulink.

6.1.1 Concept of Approximate Bisimulation

We introduce the notation for approximate bisimulation as defined in [GP11, GP07b, Gir05].

Definition 44 (Approximate Bisimulation). *Let $T_1 = (Q_1, \Sigma, \rightarrow_1, Q_1^0, \Pi, \langle\langle . \rangle\rangle_1)$ and $T_2 = (Q_2, \Sigma, \rightarrow_2, Q_2^0, \Pi, \langle\langle . \rangle\rangle_2)$ be two labelled transition systems (LTS), where Q_i are the sets of states, $Q_i^0 \subseteq Q_i$ the sets of initial states, Σ the sets of input alphabet (labels), $\rightarrow_i \subseteq Q_i \times \Sigma \times Q_i$ the transition relations, Π the set of observations and $\langle\langle . \rangle\rangle_i : Q_i \rightarrow \Pi$ mappings of states to observations ($i \in \{1, 2\}$; note that Σ and Π is in both LTS the same). Let furthermore Π be a metric space with metric $d : \Pi \times \Pi \rightarrow \mathbb{R}$. The metric allows us to measure distances in the set of observables.*

A bisimulation relation $B_\varepsilon \subseteq Q_1 \times Q_2$ of precision ε is a relation fulfilling the following properties $\forall (q_1, q_2) \in B_\varepsilon$.

1. $d(\langle\langle q_1 \rangle\rangle_1, \langle\langle q_2 \rangle\rangle_2) \leq \varepsilon$
2. $\exists q_1' \in Q_1 : q_1 \xrightarrow{\alpha} q_1' \Rightarrow \exists q_2' \in Q_2 : q_2 \xrightarrow{\alpha} q_2' \wedge (q_1', q_2') \in B_\varepsilon$
3. $\exists q_2' \in Q_2 : q_2 \xrightarrow{\alpha} q_2' \Rightarrow \exists q_1' \in Q_1 : q_1 \xrightarrow{\alpha} q_1' \wedge (q_1', q_2') \in B_\varepsilon$

The LTSs are bisimilar with precision ε, denoted as $T_1 \sim_\varepsilon T_2$, if $B_\varepsilon \subseteq Q_1 \times Q_2$ exists such that

1. *B_ε is an approximate bisimulation relation of precision ε,*
2. $\forall q_1 \in Q_1^0 \exists q_2 \in Q_2^0 : (q_1, q_2) \in B_\varepsilon$
3. $\forall q_2 \in Q_2^0 \exists q_1 \in Q_1^0 : (q_1, q_2) \in B_\varepsilon$

Note that Σ in this definition is the set of input alphabets for the LTS. It is the same symbol as we have used to denote our states in the operational semantics. We have left the symbol here as it has been introduced in [GP11, GP07b, Gir05]. In the operational semantics [BC12] to which we adapt approximate bisimulation in the next subsection, we do not have an input alphabet.

6.1.2 Adaptation of Approximate Bisimulation to Simulink

In this subsection, we adapt Definition 6.1.1 to the operational semantics of Simulink [BC12], which is our formal foundation.

To obtain a transition system suitable to instantiate the definition for the approximate bisimulation, we need to define the set of states Q, the input labels Σ, the initial states Q^0, the transition relation $\rightarrow$, the set of observations Π, and the observation function $\langle\langle . \rangle\rangle$.

Definition 45 (Transition System of Simulink)**.** *The TS for a Simulink model $M = (B, E, I, O, S, C, port, fun, p)$ is defined as*

- *$Q \subseteq \mathbb{R}^{\mathcal{V}}$ - the set of states (each variable is assigned with a value)*
- *We do not need a set of input labels Σ, i.e., $\Sigma = \{\tau\}$, where τ is a symbol indicating an empty element of the alphabet.*
- *$\xrightarrow{\tau} \subseteq Q \times Q$ is defined by the operational semantics, i.e., $\sigma \xrightarrow{\tau} \sigma'$ with $\sigma, \sigma' \in Q$ if and only if*
$$Eq, \pi \vdash \sigma \rightarrow \sigma'$$
- *$Q^0 \subseteq Q$ assigns the initial values, i.e., for stateful blocks $b \in S$, associated variable v, $\sigma(v) = init$, all other variables are initially set to 0*
- *$\Pi \subseteq (\mathbb{R} \cup \{\infty\})^V$ - the set of observations.*
- *$\langle\langle . \rangle\rangle : Q \rightarrow \Pi$, $\langle\langle \sigma \rangle\rangle(v) := \sigma(v)$ if v is a variable associated with a sink block $b \in O$, ∞ otherwise. The observation map provides the value at the sink of the model - and a symbol ∞ otherwise. This is because to apply the concept of approximate bisimulation, we are only interested in the values at points of observations, i.e., at output blocks.*

We define the metric on Π as $d_\infty : \Pi \times \Pi \rightarrow \mathbb{R}$, $d_\infty(\sigma_1, \sigma_2) := \|\sigma_1 - \sigma_2\|_\infty$ with $\|.\|_\infty : \Pi \rightarrow \mathbb{R}$, $\|\sigma\|_\infty := \max_{v \in \bigcup_{b \in O : v \text{ variable of } b}} (\sigma(v))$. It takes the largest distance of all observations.

This definition of the transition system is obviously compatible with both, the operational semantics [BC12] and the definition of the approximate bisimulation introduced in Section 6.1.1. Note that each transition system has a finite runtime yielding a specific interval length $\#\mathcal{I}$ for the evaluation, which is given by the user. The transition system contains the states evolving from each other via the inference rules of the operational semantics [BC12]. However, as soon as the end of the simulation interval has been reached, the final state does not change anymore as per the global simulation rule (cf. Section 2.3.2).

We combine the concepts in the following theorem.

Lemma 18 (Approximate Bisimulation for Simulink)**.** *Let* $M_i = (B_i, E_i, I_i, O_i, S_i, C_i, port_i, fun_i, p_i)$, $i \in \{1, 2\}$ *be two Simulink models,* $TS_i = (Q_i, \emptyset, \rightarrow_i, Q_i^0, \Pi_i, \langle\langle . \rangle\rangle_i)$ *the respective transition systems with the same user-defined finite simulation interval length* $\#\mathcal{I}$*. Let moreover* $O_1 = O_2$*, i.e., the output blocks, which are the points of observation, are the same in both models. Let* $\varepsilon \geq 0$*. Let* $B_\varepsilon \subseteq Q_1 \times Q_2$ *be a relation on the two set of states.* B_ε *is an* ***approximate bisimulation relation of precision*** ε *if* $\forall (\sigma_1, \sigma_2) \in B_\varepsilon$

1. *The maximal distance of observations is at most* ε*, i.e.,*

$$d_\infty(\langle\langle \sigma_1 \rangle\rangle_1, \langle\langle \sigma_2 \rangle\rangle_2) \leq \varepsilon$$

   A sufficient condition for this is

$$\forall b \in O : \|[\![M_1]\!]_{OS}^{\varphi}(b)(\tau) - [\![M_2]\!]_{OS}^{\varphi}(b)(\tau)\|_\infty \leq \varepsilon$$

   for an assignment φ *at a point in time* $\tau \in \mathcal{I}$ *(*$\mathcal{I}$ *is the simulation interval).* t *is determined by* $\sigma_1(t) = \sigma_2(t) = \tau$*.*

2. *If* $\exists \sigma_1' \in Q_1 : \sigma_1 \rightarrow \sigma_1'$*, then* $\exists \sigma_2' \in Q_2 : \sigma_2 \rightarrow \sigma_2' \wedge (\sigma_1', \sigma_2') \in B_\varepsilon$*. This is equivalent to*

$$\exists \sigma_1' \in Q_1 : Eq, \pi \vdash \sigma_1 \rightarrow \sigma_1' \Rightarrow \exists \sigma_2' \in Q_2 : Eq, \pi \vdash \sigma_2 \rightarrow \sigma_2' \wedge (\sigma_1', \sigma_2') \in B_\varepsilon$$

3. *If* $\exists \sigma_2' \in Q_2 : \sigma_2 \rightarrow \sigma_2'$*, then* $\exists \sigma_1' \in Q_1 : \sigma_1 \rightarrow \sigma_1' \wedge (\sigma_1', \sigma_2') \in B_\varepsilon$*. This is equivalent to*

$$\exists \sigma_2' \in Q_2 : Eq, \pi \vdash \sigma_2 \rightarrow \sigma_2' \Rightarrow \exists \sigma_1' \in Q_1 : Eq, \pi \vdash \sigma_1 \rightarrow \sigma_1' \wedge (\sigma_1', \sigma_2') \in B_\varepsilon$$

The transition systems are ***approximate bisimilar with precision*** ε*, denoted as* $T_1 \sim_\varepsilon T_2$ *if* $B_\varepsilon \subseteq Q_1 \times Q_2$ *exists such that*

1. B_ε *is an approximate bisimulation relation of precision* ε*,*
2. $\forall \sigma_1 \in Q_1^0 \exists \sigma_2 \in Q_2^0 : (\sigma_1, \sigma_2) \in B_\varepsilon$*. This means equivalently*

$$\forall b \in O : \|[\![M_1]\!]_{OS}^{\varphi}(b)(t_0) - [\![M_2]\!]_{OS}^{\varphi}(b)(t_0)\|_\infty \leq \varepsilon$$

   for an assignment φ*.* $\forall \sigma_2 \in Q_2^0 \exists \sigma_1 \in Q_1^0 : (\sigma_1, \sigma_2) \in B_\varepsilon$*, which is equivalent to the former case.*

Proof. The lemma is the instantiation of the definition of the approximate bisimulation 44 to the Simulink semantics. Recall that $[\![M]\!]_{OS}^{\varphi}(b)(\tau) = \sigma(v)$ if v is the variable

associated to b and $\sigma(t) = \tau$. Also note that the operational semantics of Simulink is deterministic. Hence, the statements in the enumeration items follow diretly from the definitions.

□

If we identify the models with their transition systems, we also briefly denote

$$M_1 \sim_{\varepsilon(\#\mathcal{I})} M_2.$$

The expression $\varepsilon(\#\mathcal{I})$ emphasises the dependency of the maximal distance ε on the user-defined finite simulation interval length $\#\mathcal{I}$. Note if $\varepsilon > 0$, the value ε generally increases with the simulation interval length. The simulation interval length is linked with the transition system, i.e., a transition system is determined via the inference rules of the operational semantics by a model together with the user-defined parameters, e.g. sample step size, approximation method, simulation start, simulation end. As soon as the simulation end has been reached, the transition system keeps the final state. Hence, our adaptation of approximate bisimulation for Simulink is sound. However, to avoid the impression that approximate bisimulation with a precision ε would imply a global threshold ε for arbitrary simulation interval lengths, we emphasise this by denoting this dependency with $\varepsilon(\#\mathcal{I})$. If it is clear from the context, we reduce the full notation $\varepsilon(\#\mathcal{I})$ to ε to keep the notation as simple as possible. Also note that for $\varepsilon = 0$, this threshold holds globally, i.e., for arbitrary lengths of simulation intervals.

Note that in Definition 45, we have only compared the values at the output blocks $b \in O$. It is also possible to compare the output of blocks within a Simulink model. We use this in various cases: The disturbance ε at control signal inputs has an impact on the switching behaviour. It may cause time-delayed switching behaviour. We consider this at the end of this Chapter. Moreover, in Chapter 7, we calculate a local ε_l within a model and propagate this to a global ε_g through subsequent submodels. The global precision ε_g is the maximal distance ε at the output blocks as in Lemma 18. If the values calculated at the output of two blocks b_1, b_2 in two models behave approximately bisimilar, we denote this by

$$[\![M_1]\!](b_1) \sim_{\varepsilon(\#\mathcal{I})} [\![M_2]\!](b_2).$$

In the next sections we link our adaptation of the approximate bisimulation with our AR. We provide sufficient conditions that enable to conclude approximate bisimulation, and means to calculate the precision ε. We discuss purely time-discrete models, purely time-continuous models, and hybrid models. The reason for us not

directly providing conditions for hybrid models is that in case of time-hybridity, we are not able to directly calculate the precision ε of the approximate bisimulation. We leave this case to future work. However, for purely time-discrete, purely time-continuous, and hybrid models with control flow, we are able to provide sufficient criteria and means to directly calculate ε. As in Chapter 5, we refer the proofs in Appendix A. However, we provide the key ideas of the proofs directly in the text.

6.2 Behavioural Equivalence for Purely Time-Discrete Models

We start with purely time-discrete models. The key idea is to establish equivalence between the solutions of the ARs of two Simulink models. This can be performed symbolically with the help of a Computer Algebra System (CAS). In the time-discrete case, we do not have an approximation. Hence, the precision ε is 0 in this case.

Theorem 4 (Behavioural Equivalence for Purely Time-Discrete Models)**.** *Let M_1 and M_2 be purely time-discrete Simulink models that are executed on the interval $\mathcal{I}$ with the same sample step size h, each time-discrete block with the same sample step size h_d. Let furthermore be $(f_n)_n, (g_n)_n$ be real-valued sequences for which the following holds for all assignments φ.*

$$(f_n)_n \models [\![M_1]\!]^\varphi, (g_n)_n \models [\![M_2]\!]^\varphi$$

If

$$\forall 0 \leq n \leq n_{max} : f_n = g_n,$$

where n_{max} is the index of the last simulation step in the simulation interval $\mathcal{I}$, then

$$M_1 \sim_0 M_2$$

This means, the models are approximately bisimilar with precision 0. In this case, the approximate bisimilarity coincides with traditional bisimulation.

For the proof, we refer to Appendix A. The key idea is that due to the uniqueness of the sequences solving the AR's equations, they yield exactly the same values in both models. Note that sequences are always uniquely determined by difference equations, i.e., if a solution exists, it is unique. This is because the difference equation is a recursive definition: Starting from the initial value, the next value in each step can be calculated by applying the arithmetic function on the previous value as defined in the difference equation. Note that the sample time steps at both models are assumed to be the same for the time-discrete blocks in both models. In Lemma

16 we have shown that the values being calculated during an evaluation of the operational semantics exactly solve the AR's equations. There is no approximation in time-discrete models. Therefore, the precision of the approximate bisimulation is 0 in this case.

Note that the sequences $(f_n)_n, (g_n)_n$ that interpret or solve the ARs $[\![M_i]\!]^\varphi$ depend on the arbitrarily chosen but fixed assignment φ. This is expressed by letting φ appear in the formula $(f_n)_n, (g_n)_n \models [\![M_i]\!]^\varphi$, and is consistent with the definitions in Section 5.2, where we have introduced the interpretations of the AR. Also note that our theorem requires the equality for all solutions $f_n = g_n$ in order to conclude approximate bisimulation, independent from the assignment. Hence, either the models are independent from assignments at all, i.e., they have no input signals, or an additional argument that allows to draw the conclusion for equality independent from the assignment φ needs to be found if we want to apply the theorem. The former leads to the idea that closed (sub-)systems, i.e., systems without input signals, can alternatively be expressed by their analytical solution, if possible. The latter leads to an application for models expressible by linear difference equations, which we describe in Chapter 7.

6.3 Behavioural Equivalence for Purely Time-Continuous Models

We proceed with purely time-continuous models. The key idea is firstly to establish equivalence of the solutions of the ARs of two given Simulink models as in the previous section. To obtain an estimate for the precision $\varepsilon(\#\mathcal{I})$ of the approximate bisimulation, we provide an estimate for differing values due to approximation during the evaluation of a given Simulink model. The sum of the estimates for source and target model in turn yields the precision $\varepsilon(\#\mathcal{I})$ of approximate bisimulation. After the main theorem, we also provide criteria for various special cases, for which our general calculation approach can be relaxed.

Theorem 5 (Approximate Behavioural Equivalence for Purely Time-Continuous Models)**.** *Let M_1 and M_2 be purely time-continuous Simulink models that are executed on the interval $\mathcal{I}$. Let furthermore be $f, g : \mathcal{I} \to \mathbb{R}$ be functions for which the following holds for all assignments φ with $Im(\varphi) \subset C(\mathcal{I}, \mathbb{R})$.*

$$f \models [\![M_1]\!]^\varphi, g \models [\![M_2]\!]^\varphi$$

If f and g are uniquely determined by the equations of the AR and

$$\forall t \in \mathcal{I} : f(t) = g(t),$$

then there exists $\varepsilon(\#\mathcal{I}) > 0$ such that

$$M_1 \sim_{\varepsilon(\#\mathcal{I})} M_2$$

For the proof, we refer to Appendix A. The key idea for the proof is the following: From Lemma 17, we know that the equations of our AR are fulfilled by a limit function emerging from sequences of semantical evaluations for decreasing sample step size. The theorem assumes that these limit functions, which are the interpretations of our ARs, are the same. This guarantees the approximate bisimulation. The precision $\varepsilon(\#\mathcal{I})$ is determined by an estimate of the approximations that occur during evaluation of the solver transitions of the operational semantics. Lemmas 14 and 15 from the Background Chapter 2 provide a global error $\varepsilon_{y_i}(\#\mathcal{I})$ ($i \in \{1, 2\}$ indicating the model M_1 or M_2) for the approximations of the state variables for both models. The values of $\varepsilon_{y_i}(\#\mathcal{I})$ are then propagated according to the output equations of the ARs to the output of the models yielding $\varepsilon_i(\#\mathcal{I})$. For instance, if the output equation is $2y$, then the value of $\varepsilon_{y_i}(\#\mathcal{I})$ is doubled. By applying the triangular equation,the sum of both $\varepsilon_i(\#\mathcal{I})$ yields the precision $\varepsilon(\#\mathcal{I})$ for the approximate bisimulation, which concludes the proof.

Note that the same remarks as in the supplement of Theorem 4 for time-discrete models apply for this theorem. In short: The functions f, g being the interpretations of the respective ARs depend on the arbitrarily chosen but fixed assignment φ. In order to conclude approximate bisimulation, i.e., in order to apply the Theorem, the user needs to guarantee equality among f and g for all assignments φ. This is either implicitly fulfilled by closed models without input signals or an aditional argument is required. We provide such an argument in Chapter 7 for a subset of Simulink models.

Note that generally, the deviation increases with increasing interval of semantical evaluation. However, since the calculation of ε depends on the sample step size, which can be modified by the user, we have control over it. The user therefore can calculate the required sample step size for a maximal deviation he or she can accept in a given execution interval. Moreover, there exist ODEs for which the error are globally bounded. This is related with stability analysis and left for future work.

Theorem 5 assumes the uniqueness of solutions of ODEs. In the following Corollary, we provide a sufficient condition for the uniqueness.

Corollary 1 (Sufficient Condition for Uniqueness of Solutions for Time-Continuous Models)**.** *Let the same assumptions hold as in Theorem 5. If the ODEs in the ARs* $[\![M_1]\!]^\varphi$ *and* $[\![M_2]\!]^\varphi$ *are globally Lipschitz continuous, then an* $\varepsilon(\#\mathcal{I})$ *exists with* $M_1 \sim_{\varepsilon(\#\mathcal{I})} M_2$*.*

Proof. Global Lipschitz continuity guarantees uniqueness of the ODE solutions. Hence, Theorem 5 is directly applicable. □

Local Lipschitz continuity also suffices for local uniqueness as we know from the Background chapter. However, in these cases, uniqueness may not be guaranteed on the whole interval $\mathcal{I}$. Further analyses are required if only local Lipschitz continuity applies, which we leave for future work.

6.3.1 Epsilon Calculation

Theorem 5 guarantees a precision $\varepsilon(\#\mathcal{I})$ for the approximate bisimulation. The key idea how to obtain such an estimate is given in the proof and supplement of the theorem. The essential information we need to capture is an estimate for the the differences of the approximation during the semantical evaluation of a given model with the mathematical solution of our AR.

According to Section 2.2.5, the approximation error for an ODE

$$\frac{d}{dt}y(t) = f(t, y(t)), y(t_0) = y_0$$

can be calculated by

$$\varepsilon_y(\#\mathcal{I}) \leq f_{err}\left(h, \#\mathcal{I}, L, \max_{t\in\mathcal{I}} \left\| \frac{d^2}{dt^2}\lambda(t) \right\|_\infty\right),$$

where f_{err} depends on the chosen approximation method (for Euler refer to Lemma 14), L is the Lipschitz constant of the right hand side of the ODE $f(t, y(t))$ for y, λ is the solution of the ODE in the AR, $\mathcal{I}$ the simulation interval, $\#\mathcal{I}$ its length. The index y in $\varepsilon_y(\#\mathcal{I})$ indicates that this is the disturbance for the stateful equations, i.e., ODEs. In other words, it is the disturbance at the output of stateful blocks. The disturbance $\varepsilon(\#\mathcal{I})$ at the output of the model depends on the output equation $out(t) = f(y(t), i(t))$, which in turn modifies $\varepsilon_y(\#\mathcal{I})$ (e.g. doubles $\varepsilon_y(\#\mathcal{I})$ if $out(t) = 2y(t)$). We clarify how we can obtain the various parameters to determine $\varepsilon_y(\#\mathcal{I})$, the approximation error of the state equation of a given model.

The value for L can be calculated by

$$L = \max_{(t,y)\in M} \|J_f(t, y)\|_\infty$$

where f is the right hand side of the ODE in the AR of the model (f needs to be differentiable). In this formula, M is a domain $[a, b] \times M'$ for an $M' \subset \mathbb{R}^n$ where f is defined. To actually calculate L, either one can find a bound for $\|J_f(t, y)\|_\infty$ globally, i.e., for $[a, b] \times \mathbb{R}^n$ or the user has additional information on the boundedness of f at his disposal. Please refer to Section 2.2.3 for background on the Jacobi matrix $J_f(t, y)$, and to Section 2.2.1 for the definition of the maximum on matrices. This follows from the Mean Value Theorem provided in the Background chapter (Lemma 6). If we apply it to all coordinate functions of y, the Jacobi Matrix provides an estimate for L. Note that the statement in the Mean Value Theorem per coordinate function is exactly the same as required for the Lipschitz condition.

The value for $\max_{a \le t \le b} \left\| \frac{d^2}{dt^2} \lambda(t) \right\|_\infty$ can be either calculated directly if λ is known, or via

$$\begin{pmatrix} \frac{\delta f_1}{\delta t}(t, y) \\ \vdots \\ \frac{\delta f_n}{\delta t}(t, y) \end{pmatrix} + \begin{pmatrix} \frac{\delta f_1}{\delta y_1}(t, y) \dots \frac{\delta f_1}{\delta y_n}(t, y) \\ \vdots \quad \vdots \\ \frac{\delta f_n}{\delta y_1}(t, y) \dots \frac{\delta f_n}{\delta y_n}(t, y) \end{pmatrix} \cdot \begin{pmatrix} f_1(t, y) \\ \vdots \\ f_n(t, y) \end{pmatrix} = f_t(t, y) + J_{f_y}(t, y) \cdot f(t, y)$$

$f_t(t, y)$ is a abbreviation for partial derivatives in t direction. This follows from Definition 23 and the Chain Rule from Lemma 5. It is

$$\frac{d^2}{dt^2}\lambda(t) = \frac{d}{dt}\frac{d}{dt}\lambda(t) = \frac{d}{dt} f(t, \lambda(t)) = J_y(t, \lambda(t)) =$$

$$f_t(t, \lambda(t)) + J_{f_y}(t, \lambda(t))\frac{d}{dt}\lambda(t) = f_t(t, \lambda(t)) + J_{f_y}(t, \lambda(t)) f(t, \lambda(t))$$

On the left hand side of the bottom line of the equation, we have split the Jacobi matrix $J_f(t, \lambda(t))$ and pulled out the first column, which is $f_t(t, \lambda(t))$. Hence, if we can obtain an estimate for

$$\|f_t(t, y) + J_{f_y}(t, y) f(t, y)\|_\infty$$

for $(t, y) \in M$, we also obtain an estimate for $\| \frac{d^2}{dt^2} \lambda(t) \|_\infty$.

In Theorem 5, we have shown the most general case: The ε is calculated as the sum of approximation errors of both given models as per triangular equation. This calculation is always possible (if the assumptions of Theorem 2 are met). However, there are some cases that allow a simpler calculation of $\varepsilon(\#\mathcal{I})$:

1. If one ODE is completely resolved, i.e., the refactoring replaces the ODE by its analytical solution, $\varepsilon(\#\mathcal{I})$ is only the approximation error of the model comprising the ODE. This is because the analytical solution is not approximated,

but it expresses the mathematical solution, which is the interpretation of the AR.

2. If the AR of two given models comprises the same ODEs, $\varepsilon = 0$. The reason is that in both cases, the same approximation technique yields the same values. Both approximations yield an error from the mathematical solution. However, they yield exactly the same error and consequently the same values. Examples for refactorings that leave the ODEs in the respective ARs untouched are an exchange of Integrator block and Gain block, or a replacement of a linear ODE by a Transfer Function block. The latter leaves the ODE unchanged because the semantics of a Transfer Function block is the same as an equivalent linear ODE as we have described in our extension of the operational semantics by Transfer Function blocks in Section 5.4.

In the next subsection, we discuss behavioural equivalence for hybrid Simulink models.

6.4 Behavioural Equivalence for Hybrid Models

In the previous sections, we have discussed approximate bisimulation for purely time-discrete and purely time-continuous models. In this subsection, we extend this to hybrid Simulink models with control flow. We start with a brief subsection for time-hybrid models. The aim of this is to justify our decision for excluding such models in this thesis and leave it for future work. The main reason is that it is difficult to automatically calculate a disturbance ε of the approximation error in the time-hybrid case. After the brief discussion of time-hybrid models, we proceed with models with control flow elements.

6.4.1 Time-Hybrid Models

As introduced in Chapter 5, time-hybrid models are such models whose AR contains ODEs that depend on time-discrete state variables, and vice versa. Such equations are also referred to as Delay Differential Equations (DDEs) because ODEs contain elements that refer to values from preceding time-steps, i.e., delays. In Section 4.1, we have stated that in general, we do not support such models. In this subsection, we briefly provide reasons for this decision.

Consider, we have a time-hybrid model M_1 and a refactored counterpart M_2. From Chapter 5, we know that the AR consists of an output equation, and state

equations. The state equations in the time-hybrid case are of the form

$$\frac{d}{dt}\boldsymbol{y}_c(t) = f_c(\boldsymbol{i}(t), \boldsymbol{y}_c(t), \boldsymbol{y}_d), \boldsymbol{y}_{d_{n+1}} = f_d(\boldsymbol{i}_n, \boldsymbol{y}_c(t_n), \boldsymbol{y}_{d_n}).$$

We omit the initial values here for simplicity. The input variables are replaced by the assignment, which yields

$$\frac{d}{dt}\boldsymbol{y}_c(t) = f_c(t, \boldsymbol{y}_c(t), \boldsymbol{y}_d), \boldsymbol{y}_{d_{n+1}} = f_d(t_n, \boldsymbol{y}_c(t_n), \boldsymbol{y}_{d_n}).$$

The index c stands for continuous, d for discrete. An interpretation is an interval-wise defined function that solves the interval-wise ODEs

$$\frac{d}{dt}\boldsymbol{y}_c(t) = f_c(t, \boldsymbol{y}_c(t), f_d(t_n, \boldsymbol{y}_c(t_n), \boldsymbol{y}_d(t_n))) \text{ for } t \in [t_{n+1}, t_{n+2}[.$$

At time steps $t_n := t_0 + n \cdot h_d$, the time-discrete blocks are evaluated. This causes a change of the ODE for the subsequent interval, which in turn determines the values for the next update of the time-discrete variables.

As discussed in the previous section, it is key to obtain an estimate for the approximation error during semantical evaluation of the models in order to obtain the precision $\varepsilon(\#\mathcal{I})$ of the approximate bisimulation. However, the above discussion illustrates why this is in general difficult to obtain. There are approaches available that deal with this topic, e.g. [Arn84]. However, we leave the time-hybrid cases to future work.

However, in our automation approach in Chapter 7, we allow systems where the time-discrete and time-continuous parts do not appear in the same feedback loop. In these cases, the ODE may depend on time-discrete state variables, but not the other way around. Another possibility we support is that a difference equation may depend on time-continuous state variables. But again, not the other way around. Such models allow a partitioning into purely time-continuous, and time-discrete model parts, which in turn can be treated as already discussed. We refer the details to Chapter 7.

In the next subsection, we discuss hybrid models with control flow.

6.4.2 Hybrid Models

We now proceed with models with control flow elements that are not time-hybrid, i.e., delay differential equations do not occur in the AR (cf. Definition 39). We start by discussing an issue that may occur if refactorings yield a disturbance $\varepsilon > 0$ impacting control signals. In such cases, due to the disturbance, a time-delayed

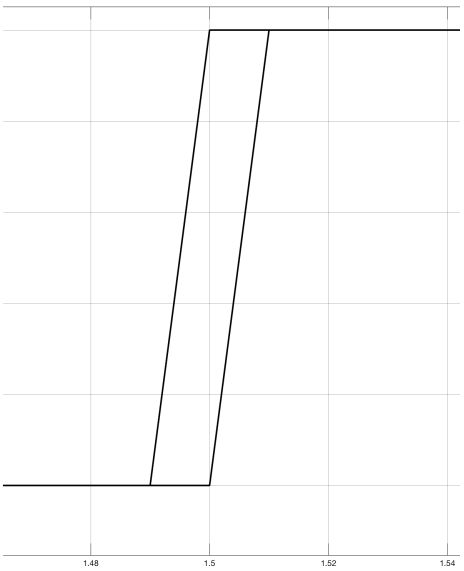

FIGURE 6.1: Time-Delayed Behaviour in Refactored Hybrid Models with Control Flow

switching behaviour may occur. We provide conditions to preclude such time delays or to identify time intervals in which such behaviour may occur. After that, we provide the theorem allowing to conclude approximate bisimulation for hybrid models with control flow together with the precision $\varepsilon(\#\mathcal{I})$ of the approximate bisimulation, which is the maximum of all $\varepsilon_\chi(\#\mathcal{I})$ from all data flow models. We have published this approach in [SHGG16a].

Time-Delayed Switching

If a disturbance due to a refactoring impacts a control signal at a control flow element, there may be a time-delay during the switch compared between source and refactored target model. Figure 6.1 provides an example. There, two graphs are depicted. One showing the trajectory of the source model, the other of the target model. A switch occurs. The figure exhibits a time delay at the switch. The cause is a disturbance $\varepsilon_c > 0$ at a control flow element.

In this subsection, we provide conditions that allow to preclude such unwanted time-delays, or to isolate time intervals in which a time-delay may occur.

Before we come to the conditions, we define the phenomenon of a time-delayed switch.

Definition 46 (Time-Delayed Switch)**.** *Let M_1, M_2 be two hybrid Simulink models with the same set of configurations X, evaluated in simulation interval $\mathcal{I}$ with the same sample step size. Moreover, let*

$$\forall \chi \in X : f_{1_{control_\chi}} = f_{2_{control_\chi}} \text{ for } f_{i_{control_\chi}} \models [\![M_{i_\chi}]\!]^\varphi(b_{control})$$

There, φ is an assignment. We have assembled all control signals of the configuration χ in a vector, abbreviated $f_{i_{control_\chi}} \models [\![M_{i_\chi}]\!]^\varphi(b_{control})$. It means, the ARs of the control signals determining the switching times yield the same values. Consider, there is a switch in M_1, i.e., a change from a configuration χ_1 to a configuration χ_2 in M_1. This happens if the fulfillment of a condition at at least one of the control flow elements changes at a time step $t_s \in \mathcal{I}$.

1. *If the change in M_2 from χ_1 to χ_2 happens at the same time step t_s, we call this a **perfectly synchronous switch**.*
2. *It the change in M_2 from χ_1 to χ_2 happen at a later step $t_{s'}$, then we call this a **time-delayed switch**. The distance $t_{s'} - t_s$ is the delay.*

For time-discrete models that fulfill the conditions of Theorem 4, the semantical evaluation of both models yield the same values, particularly at all control flow elements. Hence, the switching is always perfectly synchronous for time-discrete models.The following conditions therefore only consider time-continuous submodels impacting the control signal. We exclude time-hybrid cases here as discussed in the previous subsection. They are part of future work.

We provide each of the conditions for one control flow element. Since the timing of switches, i.e., the time step of occurring changes between configurations in a model, are determined by the conjunction of all conditions at all control flow elements, the statements simply need to be applied to each control flow elements to obtain a statement for the overall model.

We discuss two conditions. The first one is that control signals including the disturbance ε_c at the control signal is too far away from the threshold to be able to hit it, or a crossing of a control signal including the disturbance happens between two sample steps. In this case, no switch happens in both models. The second condition is that a crossing happens in both models. If the crossing of control signal in both models happens between sample steps, the switch is perfectly synchronous. Otherwise, the condition can be used to determine intervals where time-delayed behaviour can occur.

We start with the first condition: Let M_1 and M_2 be hybrid Simulink models with control flow containing a single Switch block, i.e., $C_i = \{b_{switch}\}$. Without loss

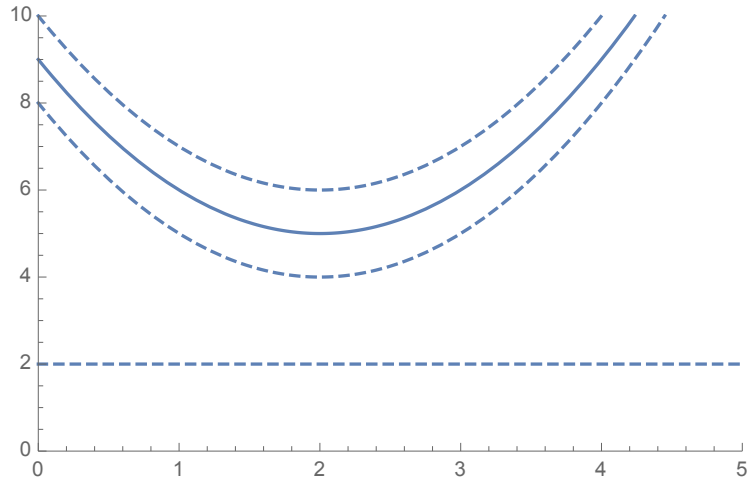

FIGURE 6.2: First Timing Condition: No crossing

of generality, we assume that the condition associated with the Switch block is $> \xi$ for a real value ξ, i.e., $p(b_{switch}) > \xi$. Let the simulation interval be $\mathcal{I}$. Let

$$\lambda_{1_{control_j}} = \lambda_{2_{control_j}} \text{ for } \lambda_{i_{control_j}} \models [\![M_{i_{\chi_j}}]\!]^{\varphi}(b_{control})$$

Let $\varepsilon_c := \max_{i \in \{1,2\}} \varepsilon_{c_i}$ with ε_{c_i} the numbers fulfilling

$$[\![M_{1_i}]\!]^{\varphi}(b_{control}) \sim_{\varepsilon_{c_i}} [\![M_{2_i}]\!]^{\varphi}(b_{control}).$$

The ε_{c_i} are the precisions of the approximate bisimulations at the control port of the Switch block for the configurations $i \in \{1, 2\}$. They are guaranteed by Theorem 4 for time-discrete submodels (in this case, $\varepsilon_c = 0$, which always results in perfectly synchronous switching as discussed), and Theorem 5 for time-continuous submodels.

For $\varepsilon_c > 0$, no switch occurs, if

$$\forall t \in \mathcal{I} : \lambda_c(t) - \xi > \varepsilon_c.$$

Figure 6.2 illustrates the condition: The curve in the middle of the figure is the trajectory of the control signal λ_c, the dashed lines around it depict the ε_c tube around it. The threshold at the control flow element is depicted as the dashed line. The condition expresses that it is impossible for the trajectory including the disturbance ε_c to cross the threshold. Hence, no switch occurs in both models. The second condition (not depicted) treats with the negative case (then, the condition $> \xi$ is never fulfilled).

The next condition also concerns with the case that no switch occurs in both models. It relaxes the first condition: The trajectory including ε_c may cross the threshold of the control flow element. However, the crossing happens between two sample steps. More formally, consider

$$zero_1(t) := \lambda_{c_1}(t) - \varepsilon_c - \xi, zero_2(t) := \xi - \lambda_{c_2}(t) + \varepsilon_c.$$

It is the translation of the trajectory including the disturbance and the threshold to the x-axis. Moreover, consider the zero crossings of these functions

$$\mathcal{Z}_i := zero_i^{-1}(0) = \{t \in \mathcal{I} | zero_i(t) = 0\}.$$

If $zero_1(t_0) > 0$ and $\mathcal{Z}_1 = \{t_1, ..., t_n\}$ is a finite, discrete set (We have defined discrete sets in Section 2.2.1) with an even amount of elements and without loss of generality ordered, i.e., $\forall 1 \leq i \leq n : t_i < t_{i+1}$, then no switch occurs if

$$\forall 1 \leq i \leq n : n \text{ odd } \forall t \in S : t \notin [t_i, t_{i+1}]$$

This means, that all zero crossings happen between time steps and are therefore undetected in both models. The set S is the set of sample time steps. Figure 6.3 illustrates this case. There, the sample steps are depicted by dashed parallels to the y-axis. The crossing of the threshold happens between sample steps. For $zero_2(t_0) < 0$, an analogous condition with $\mathcal{Z}_2$ can be formulated.The same is the case for perfect sets, i.e., if the zero crossings are not discrete but remain for an interval on the x-axis. Then, this interval must as well be between two sample steps in order to be undetected.

The last condition concerns with detection of intervals where time-delayed behaviour may occur. This may happen if the conditions above cannot be guaranteed. In this case, the ordered zero crossings of the functions

$$zero_-^+ := \lambda_{c_1}(t) - \varepsilon_c - \xi, zero_+^+ := \lambda_{c_1}(t) + \varepsilon_c - \xi,$$

$$zero_-^- := \xi - \lambda_{c_2} - \varepsilon_c, zero_+^- := \xi - \lambda_{c_2} + \varepsilon_c$$

yield intervals during which time steps occur. These time steps are those where time-delayed behaviour may occur. Figure 6.4 illustrates this. There, the trajectory of the control signal crosses the threshold. The ε_{c_1} tube surrounding the trajectory crosses the threshold left and right from it. The sample steps between the interval (depicted as vertical dashed lines) are those where time-delayed behaviour may occur.

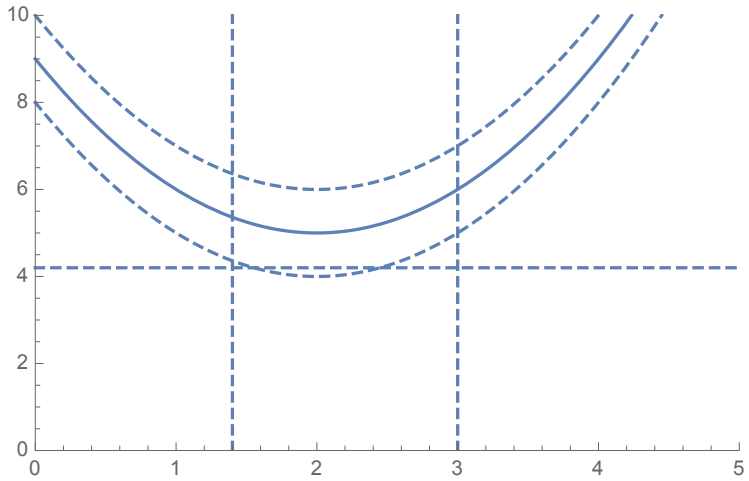

FIGURE 6.3: Second Timing Condition: Undetected Crossing

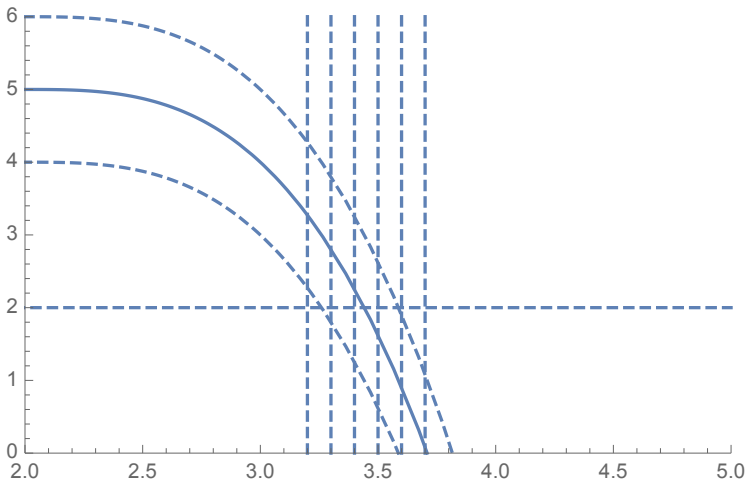

FIGURE 6.4: Intervals of Potentially Asynchronous Switching

All functions in the above conditions can be analysed with a Computer Algebra System (CAS) if the trajectory, i.e., the solution of the AR for the control signals can be analytically solved. Then, the CAS can be used to determine potential intervals for time-delayed behaviour.

Behavioural Equivalence of Hybrid Models with Control Flow

In this last subsection, we provide the main theorem that allows to conclude approximate bisimulation for hybrid Simulink models.

Theorem 6 (Approximate Bisimulation for Hybrid Models)**.** *Let M_1, M_2 be two hybrid Simulink models with the same set of configurations X, evaluated in simulation interval $\mathcal{I}$ with the same sample step size. Moreover, for all assignments φ, let*

$$\forall \chi \in X : f_{1_\chi} = f_{2_\chi} \text{ for } f_{i_\chi} \models [\![M_{i_\chi}]\!]^\varphi.$$

This means, the solutions of our AR yield all the same values for all output signals including the control signals. Let all state equations in the AR be uniquely and globally solvable (which is for difference equations always the case, for differential equations fulfilled if the right hand sides of the equations are globally Lipschitz continuous). Let all data flow models be purely time-discrete or purely time-continuous, i.e., no data flow model time-hybrid. Moreover, let

$$\forall \chi \in X : cond_1(\chi) = cond_2(\chi).$$

This means, all conditions associated with the configurations are the same. Let $\varepsilon_\chi(\#\mathcal{I})$ be the numbers such that

$$M_{1_\chi} \sim_{\varepsilon_\chi(\#\mathcal{I})} M_{2_\chi},$$

i.e., $\varepsilon_\chi(\#\mathcal{I})$ are the precisions of the approximate bisimulation for the data flow models with the same configurations. Then

$$M_1 \sim_{\varepsilon(\#\mathcal{I})} M_2$$

with

$$\varepsilon(\#\mathcal{I}) := \max_{\chi \in X} \varepsilon_\chi(\#\mathcal{I})$$

if no time-delayed switching as discussed in the previous section occurs. Otherwise, $\varepsilon(\#\mathcal{I}) = \max_{\chi \in X} \varepsilon_\chi(\#\mathcal{I}) + \max_{t \in \mathcal{I}, \chi \in X} \|f_{1_\chi} - f_{2_\chi}\|_\infty$.

For the proof, we refer to Appendix A. The key idea is that the assumption in the theorem ensures that trajectories of all data flow models yield the same values. Since we have also declared the blocks at control ports of control flow elements to be output ports in our definition of the AR for hybrid models, this includes the

equality of the trajectories for the control signals. Control signals together with the conditions of control flow elements determine the time step at which switches between configurations occur. The equality ensures that these time steps are the same, unless time-delayed switching has been identified as described in the previous subsection. The global precision $\varepsilon(\#\mathcal{I})$ is the maximum of all possible disturbances $\varepsilon_\chi(\#\mathcal{I})$ for all configurations, i.e., all disturbances coming from a possible data flow through the model.

6.5 Summary

In this chapter, we have presented our adaption of the concept of approximate bisimulation to Simulink. With this, we are able to address the approximate nature of signal flow-oriented modelling languages such as Simulink. Approximate bisimulation is an established equivalence notion to compare trajectories that are defined using an operational semantics with respect to a precision ε. Our adaptation connects this equivalence notion with our AR. Moreover, we have provided proof obligations, i.e., sufficient conditions for approximate bisimulation of source and refactored target model. These conditions operate on the denotational level of the ARs. Our approach includes the calculation of the precision ε of the approximate bisimulation.

In the next chapter, we describe the automation of our approach for a relevant subset of Simulink models.

Chapter 7

Automated Equivalence Checking for Hybrid Control Systems

In this chapter, we describe the automation of our verification methodology. In order to obtain a largely automated solution, we restrict the set of supported Simulink models, mainly to those that can be described by linear ODEs or difference equations, which is fulfilled for many applications. A detailed list of our assumptions and limitations can be found in Section 4.1.

The key ideas are the following.

1. We provide a partitioning of each data flow model. We demand the partitions to be purely time-discrete or purely time-continuous. However, the type may change between partitions. This allows to apply our methodology even for a subset of time-hybrid models. Moreover, our partitioning splits the model into smaller components and therefore allows a modular verification.

2. In order to apply Theorems 4, 5, and 6, we perform verification of symbolic equivalence with a Computer Algebra System. In order to obtain a largely automated symbolic equivalence check, we make use of Laplace transform and z-transform of the ARs. These transformations translate complex ODEs into algebraic expressions. Symbolic equivalence can automatically be checked for algebraic expressions.

3. After having established symbolic equivalence, we calculate the precision $\varepsilon(\#\mathcal{I})$ of approximate bisimulation. Theorems 4, 5, and 6 provide the means to calculate them for models where data flow models are purely time-discrete or time-continuous. To support the described subset of time-hybrid models, we firstly apply the theorems to calculate a *local* precision $\varepsilon_{\chi_l}(\#\mathcal{I})$ (M_{i_χ} ($i \in \{1, 2\}$) being the respective data flow model for models M_1, M_2) at submodels exchanging ODEs by analytical solutions, and secondly we provide a propagation mechanism to calculate the *global* precision $\varepsilon_{\chi_g}(\#\mathcal{I})$ for the overall

data flow models M_{i_χ}. The data flow models are approximately bisimlar with precision $\varepsilon_{\chi_g}(\#\mathcal{I})$. As per theorem 6, the maximum of the $\varepsilon_{\chi_g}(\#\mathcal{I})$ of all data flow models yields the precision $\varepsilon(\#\mathcal{I})$ of the overall model.

With our automated approach, we are able to verify the correctness of basically the following refactoring types.

1. All kinds of syntactic refactorings, such as changes in subsystem or bus hierarchy.
2. All kinds of arithmetic refactorings, such as exploiting distributivity law, exchange of Integrator and Gain blocks.
3. Exchanges of submodels representing ODEs or difference equations by (Discrete) Transfer Function blocks or vice versa.
4. Exchanges of submodels representing ODEs or difference equations by submodels representing their analytic solutions or vice versa.

In essence, we are able to verify the state of the art refactorings that are currently performed without verification, mostly manual. These are represented by items (1) and (2). Moreover, we are able to verify correctness of important semantical refactorings, which are represented by items (3) and (4).

In this chapter, we describe the elements of our automation approach in detail. We go through each step, and discuss the soundness directly in the steps. We start with guarantees the user needs to provide in the next section. These guarantees ensure the applicability of our methodology. We continue with the data flow model calculation and the partitioning in Section 7.2. Then, we present symbolic equivalence checking in Section 7.3. We continue with the epsilon calculation and propagation in Section 7.4. In Section 7.5, we combine the results of all steps, apply 6 to conclude approximate bisimulation of precision $\varepsilon(\#\mathcal{I})$ for two given Simulink models fulfilling our limitations, and therefore directly discuss the soundness of our overall approach there.

7.1 Obligations to ensure Applicability of our Methodology

To ensure applicability of our methodology, some prerequisites must be fulfilled. These cannot be checked. Therefore, the user is asked to ensure them.

1. As described in Chapter 5, in Lemma 17, for the soundness of our AR in the time-continuous case, it is required that the we have continuous, piecewise bounded values at stateful blocks. Due to the linearity of our subset of

supported models, this reduces to the requirement that ingoing signals fulfill these properties. The user is asked to guarantee that.

2. In our approach, we conclude the equality of the solutions of the extracted ARs of source and target model from the equality of the Laplace transformed or z-transformed expressions. This demands injectivity of Laplace transform or z-transform. According to Chapter 2, to guarantee injectivity of Laplace transform or z-transform, the ingoing signals of the model need to be piecewise-smooth or continuous and of at most exponential growth. Both conditions need to be confirmed by the user.

3. The user is asked to guarantee the equality of the ingoing signals of source and target model.

4. The user is asked to confirm that in source and target models the same approximation method is selected.

The conditions are fulfilled for all practical applications: Signals that are unbounded or grow faster than the exponential function do not play important roles in practice.

7.2 Data Flow Model Calculation and Partitioning

We start with data flow model calculation and partitioning. Since both are on a syntactical level, we discuss them together as syntactical preparations. Data flow models represent the possible data flows in the model as discussed in Chapter 5. The union of the ARs of all data flow models together with the equations and conditions at control flow elements yields the AR of the overall model. As discussed in Theorem 6, we verify equivalence for all data flow models in order to obtain equivalence of the overall model. The purpose of partitioning is to prepare a modular verification, and to allow alternating types of Simulink submodels: partitions containing purely time-discrete submodels can follow partitions containing purely time-continuous submodels and vice versa. A partition represents an undivisible submodel: Since a partition is purely time-discrete or purely time-continuous, and perceivable as a Simulink model with ingoing signals from preceding partitions, and outgoing signals to subsequent partitions in the partitioning, our Theorems 4 for behavioural equivalence of purely time-discrete Simulink models and 5 for behavioural equivalence of purely time-continuous Simulink models are applicable.

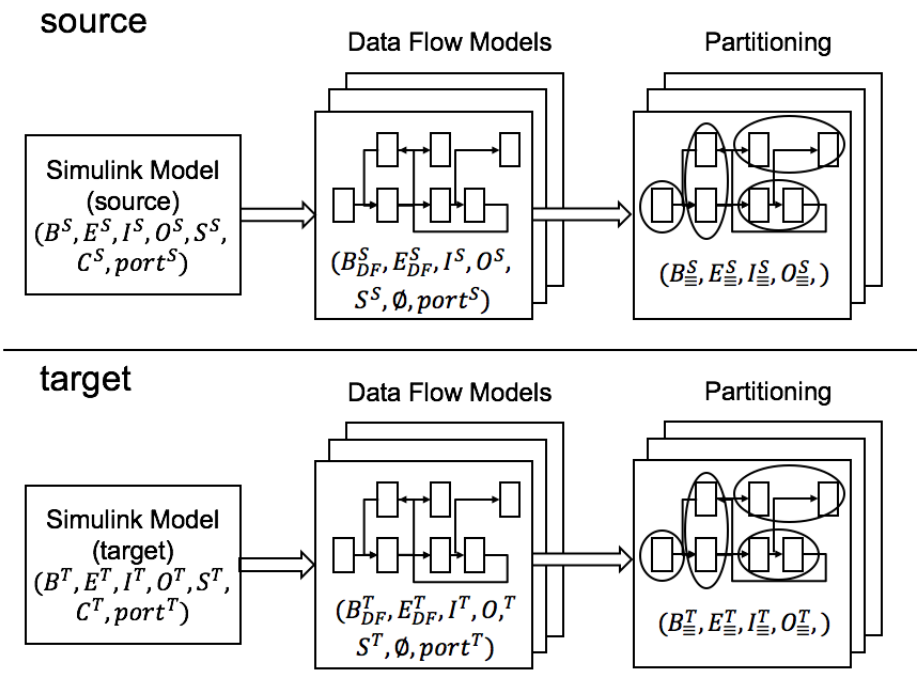

FIGURE 7.1: Syntactical Preparations Overview

Figure 7.1 provides an overview over these steps. The figure should be read from left to right. At the top, there is the source model, at the bottom the target model. So, all steps are performed for both models.In the following, we describe both steps of the syntactical preparations.

7.2.1 Data Flow Model Calculation

We start with the calculation of data flow models. Data flow models are associated with configurations. We have defined them in Definition 37. This definition provides a precise description what we perform to obtain the data flow models.

In particular, we perform the following steps for each configuration, i.e., each $n-$tuple $(\chi_1, ..., \chi_n)$, n being the amount of Switch blocks, $\chi_i \in \{1, 2\}$ for all Switch blocks s_i:

1. At Switch block s_i, we cut all edges to the block, i.e., the edges at ports 1 to 3
2. We newly create an output block $b_{control_i}$ and connect it with the block that carries the control signal for s_i.
3. We connect the block that precedes s_i at data port χ_i with all successors of the Switch block s_i.

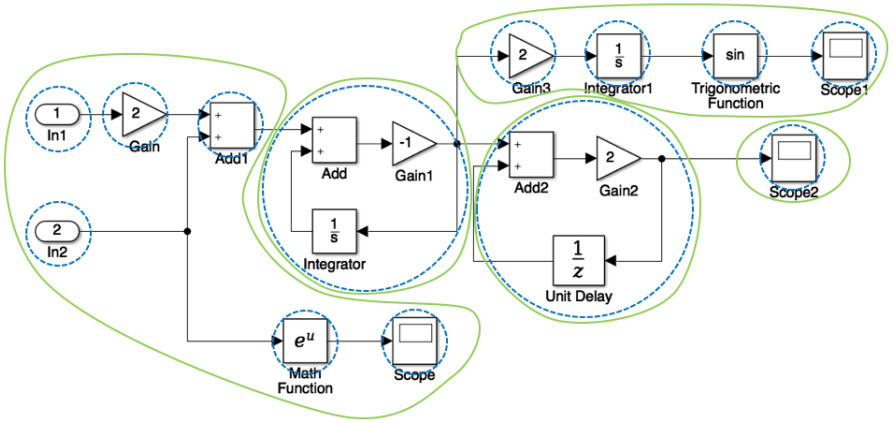

FIGURE 7.2: Partitioning Principle

4. We remove the Switch block s_i.

We also keep the predicate of all Switch blocks for the respective configuration, i.e., we keep $p_\chi(b_{control_i}) = \diamond b_{control_i} p(s_i)$ as defined in Definition 37.

The process yields 2^n data flow models.

7.2.2 Partitioning

Partitioning runs on the data flow models that we have calculated in the previous step. Partitions in the partitioning are Simulink models with ingoing signals from preceding partitions in the partitioning, and ougoing signals to subsequent partitions of the partitioning. They represent indivisible submodels, whose equivalence can be verified separately.

In the partitioning process step, we divide the data flow models in components whose equivalence can be verified separately in a subsequent step of our approach. Partitioning fulfills chiefly two objectives: 1) The verification becomes modular and therefore performance of the verification is improved, 2) the submodel types between partitions can change, i.e., time-discrete submodels can follow time-continuous submodels or vice versa. After having performed the partitioning, we enrich the partitions with output and input blocks. This allows us to treat each partition in the partitioning as a submodel. For these submodels, we can apply Theorems 4 for behavioural equivalence of purely time-discrete Simulink models and 5 for behavioural equivalence of purely time-continuous Simulink models later in the verification process.

The partitioning process is threefold. In the first step, we give the user the opportunity to mark partitions that are considered to be equivalent. In particular,

the user is required to tell the system which submodel in the original model has been replaced by its analytical solution.

In the second step of the partitioning process, we identify cyclic components, i.e., we collect strongly connected components to detect feedback loops. The idea is that in order to obtain applicability of Theorem 4 or Theorem 5 to show behavioural equivalence, we require cyclic submodels to be purely time-discrete or purely time-continuous. If both types occurred in a cycle, this would yield a time-hybrid system, which we have left for future work. Hence, cyclic submodels are indivisible in our approach and therefore we collect strongly connected components in the Simulink graph representation.

In the third step, we collect the non-cyclic components, i.e., singleton partitions from the previous step together. As for the partitions containing strongly connected components of the Simulink graph, we consider the directed acyclic graph parts (DAG parts) as indivisible components that are either purely time-discrete or purely time-continuous. Note that if these DAG parts are time-hybrid, an additional separation into purely time-discrete or purely time-continuous parts would be possible. However, we have omitted this here. An adaptation to support this would easily be possible.

Figure 7.2 illustrates the concept of the second and third step. There, the blue, dotted circles are the result of the second step in the given example. Every strongly connected component is gathered in a partition. The green, straight circles depict the result of the second step of the partitioning process. Partitions with more than one element are kept, singleton partitions are collected together if they are connected.

In the following, we go through the details of each step and formalise the ideas.

First Partitioning Step - User-Defined Partitioning

With the user-defined partitioning, the user defines where an ODE in a given model has been replaced by its analytical solution in the target model. The user provides a list of pairs of blocks, one from the source, one from the target model. These blocks together with all preceding blocks with respect to the Simulink graph are considered to be in a partition.

The motivation for this step is that the resolution of a submodel representing an ODE by its analytical solution may imply the resolution of a cycle to a DAG component, i.e., in the target model, the cycle has gone. Figure 7.3 illustrates this. There, the ODE at the beginning has been reduced by a Sine Wave generator. This Sine Wave Generator is part of a larger DAG component. The other partitioning

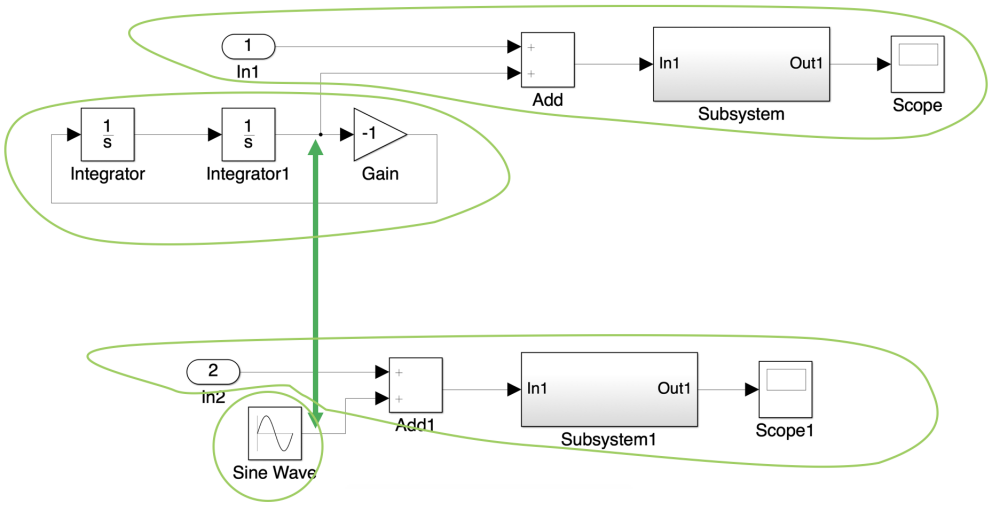

FIGURE 7.3: Uder-Defined Partitioning Example

steps, i.e., cycle detection and DAG parts assembly, would yield differing partitions for which a symbolic verification would fail.

We collect all blocks preceding the user-defined pair of blocks because replacements of submodels by analytical solutions can only occur at the beginning of the models with respect to the data flow. This becomes clear because of the meaning of a replacement of a submodel by its analytical solution. It means, a model expressing an ODE $\frac{d}{dt}\boldsymbol{y}(t) = f(t, \boldsymbol{y})$ is replaced by a model expressing its solution $\boldsymbol{\varphi}(t)$ (the time-discrete case with difference equations works analogously). The solution does only depend on t. No further dependence, i.e., on stateful blocks, is possible. Moreover, the expression $f(t, y)$ does only depend on t and the stateful variables $\boldsymbol{y}$, not on arbitrary input variables. It would not be possible to provide an analytical solution for unknown arbitrary input signals. This implies that such cases only may occur at the beginning of the model with respect to the data flow or graph structure. Hence, a partitioning combining all components together at the beginning is the only reasonable possibility.

The algorithm that performs this first step is straightforward: Start with the collection of the partition at the block defined by the user, move backwards in the graph, i.e., along E^{-1}, collect the blocks you encounter if not visited before. This yields a partition (or multiple if the user defined multiple blocks) at the beginning of the model. The partitions resulting from the first step are never modified by subsequent partitioning steps.

Second Partitioning Step - Cycle Partitioning

The next partitioning step is the cycle detection and collection of cyclic components in partitions. The result is a partitioning comprising partitions with cyclic graph components and singleton partitions with elements that are not part of a cycle. For the realisation of this step, we make use of the concept of strongly connected components in a graph [CLRS01].

Definition 47 (Strongly Connected Components)**.** *Let $G = (V, E)$ be a directed graph with the finite set V of vertices and $E \subseteq V \times V$ the set of edges. A **path** in G from $u \in V$ to $v \in V$ is a sequence $v_1, ..., v_n$ of vertices where $\forall 1 \leq i < n : (v_i, v_{i+1}) \in E$, i.e., succeeding elements in the sequence are connected in the graph, and $u = v_1, v = v_n$. G is called **strongly connected** if for every pair $u, v \in V$ there is a path from u to v and vice versa. The **strongly connected components** of G are the induced equivalence classes of the equivalence relation*

$$u \equiv v \Leftrightarrow \textit{ exists a path from } u \textit{ to } v \textit{ and vice versa}$$

The result of the partitioning of a data flow model $M = (B, E, I, O, S, \emptyset, port, fun, \emptyset)$ is the graph $M_{\equiv} = (B_{\equiv}, E_{\equiv}, I_{\equiv}, O_{\equiv})$ with

- $B_{\equiv} = B/\equiv$, i.e., $B_{\equiv}$ being the set of equivalence classes of the relation $\equiv \subseteq B \times B$.
- $E_{\equiv} \subseteq B_{\equiv} \times B_{\equiv}$ with

  $$E_{\equiv} := \{(p_1, p_2) | p_1, p_2 \in B_{\equiv} \wedge p_1 \neq p_2 \wedge \exists b_1 \in p_1, b_2 \in p_2 : (b_1, b_2) \in E\}$$

  This means, $E_{\equiv}$ connects different partitions that contain elements being connected in the original graph. We omit self-references, i.e., elements (p, p) in $E_{\equiv}$. $E_{\equiv}$ lifts the connections existing in the graph to the partition level, This means on the one hand that if in the graph two vertices are connected that are located in different partitions, we have a connection in $E_{\equiv}$. On the other hand side, if we have a connection in $E_{\equiv}$, then there is at least a connection in the underlying graph between vertices located in the respective partitions.
- $I_{\equiv} \subset B_{\equiv}$ with

  $$I_{\equiv} = \{p | \exists b \in p \text{ that is an input block}\}$$

  Note that such partitions are always singleton partitions, i.e., partitions containing only one element because input vertices can never be part of a cycle.
- $O_{\equiv}$ is defined analogously as $I_{\equiv}$ for output blocks.

This yields the blue, dotted circles in the example depicted in Figure 7.2. The elements of $E_{\equiv}$ are the connections between the partitions.

Third Partitioning Step - Merging Singleton Partitions

In the third step, the singleton partitions, i.e., partitions containing a single element, are merged. The result of this step are the plain circles depicted in Figure 7.2. Cyclic partitions, (Discrete) Transfer Function blocks, and user-defined partitions are not modified. They remain unchanged in this step. (Discrete) Transfer Functions are not merged because we consider them also as cycles. This is because they can be resolved to cyclic ODEs as described in Section 5.4.

The algorithm is as follows:

- We start at output ports of the model, i.e., at elements $b \in O_{\equiv}$, and at singleton partitions preceding a cycle or a singleton partition containing a (Discrete) Transfer Function block, i.e., at $p \in B_{\equiv}$ such that $\exists p' \in B_{\equiv} : (p, p') \in E_{\equiv}$ and $\#p' > 1$ or $\#p = 1$ and p contains a (Discrete) Transfer Function block.
- We move backwards in the partitioning, i.e., along $E_{\equiv}^{-1}$, and merge all singleton partitions.
- We stop at cyclic partitions, user-defined partitions (result of step 1), singleton partitions containing (Discrete) Transfer Function blocks, and at partitions containing input blocks.

Note that the resulting partitioning can introduce new cycles. In this case, such cycles are merged as well. Note that the reachability relation in our partitioning imposes a partial order because it is a directed acyclic graph.

Definition 48. *Let $M = (B, E, I, O, S, \emptyset, port, fun, \emptyset)$ be a data flow model.We call the result of the three algorithmic steps as described above* Partitioning *of the data flow model and denote it with*

$$(\mathfrak{P}, \mathfrak{E})$$

The set $\mathfrak{P} \subseteq \mathcal{P}(B)$ conists of subsets of vertices from the original Simulink model graph. The set $\mathfrak{E}$ connects the elements of $\mathfrak{P}$. A connection is in $E_{\mathfrak{P}}$ if and only if there is a connection in E, i.e., in the original graph, between two vertices being part of different partitions in $\mathfrak{P}$. Hence, $E_{\mathfrak{P}}$ is analogously defined to $E_{\equiv}$.

Note that the partitioning is a partitioning in mathematical sense, i.e., partitions are pairwise disjoint, and the union of all partitions yield the whole set of vertices.

To prepare the subsequent process step, the symbolic equivalence check, we add new distinguished vertices per partition, the *interface vertices*. The interface vertices are considered as new input and output vertices for the respective partition. In our verification, partitions are considered as Simulink submodels with ingoing signals from preceding partitions in the partitioning, and outgoing signals to subsequent partitions in the partitioning. The interface vertices complete the submodels in the partitions to full Simulink models that can be verified separately. We call the result Enriched Partitioning, which we define as follows.

Definition 49 (Enriched Partitioning)**.** *Let $M = (B, E, I, O, S, C, port, fun, p)$ be a Simulink model, M_χ a data flow model, $(\mathfrak{P}, \mathfrak{E})$ the result of the Partitioning process of M_χ, and $P \in \mathfrak{P}$ a partition. We identify the following interface edges in P:*

$$E_P^I := \{(b', b) \in E | \exists P' \in \mathfrak{P} : b' \in P' \wedge (P', P) \in \mathfrak{E}\}$$

$$E_P^O := \{(b, b') \in E | \exists P' \in \mathfrak{P} : b' \in P' \wedge (P, P') \in \mathfrak{E}\}$$

*The set E_P^I contains the edges imposed by the graph structure of the original Simulink model crossing adjacent partitions and **ending** in a vertex of the respective partition P. The set E_P^O contains the edges crossing adjacent partitions and **originating** from a vertex of partition P. For each of these edges in the sets E_P^I and E_P^O, we add an auxiliary vertex. We define the following sets:*

$$I_P := \{b | \exists (b_1, b_2) \in E_P^I \vee b \in I \cap P\}, O_P := \{b | \exists (b_1, b_2) \in E_P^O \vee b \in O \cap P\}$$

*We call the elements of I_P **Interface Input Vertices** of P. The elements of the set O_P are the **Interface Output Vertices** of the partition P respectively. Interface vertices consist of the newly created vertices at partition-crossing edges, and the input and output blocks of the model. We ensure that the interface vertices are properly connected in the graph, i.e., we form*

$$E_P := \{(b_I, b) | b_I \in I_P \wedge (b', b) \in E_P^I\} \cup \{(b, b_O) | b_O \in O_P \wedge (b, b') \in E_P^O\}$$

We assemble all interface input and output vertices in sets.

$$\mathfrak{I} := \bigcup_{P \in \mathfrak{P}} I_P, \mathfrak{O} := \bigcup_{P \in \mathfrak{P}} O_P$$

This yields a new data flow model with vertices $B \cup \mathfrak{I} \cup \mathfrak{O}$ and edges $E \cup \bigcup_{P \in \mathfrak{P}} E_P$. We also obtain a new Partitioning by adding the vertices of I_P and O_P to the respective partition P.

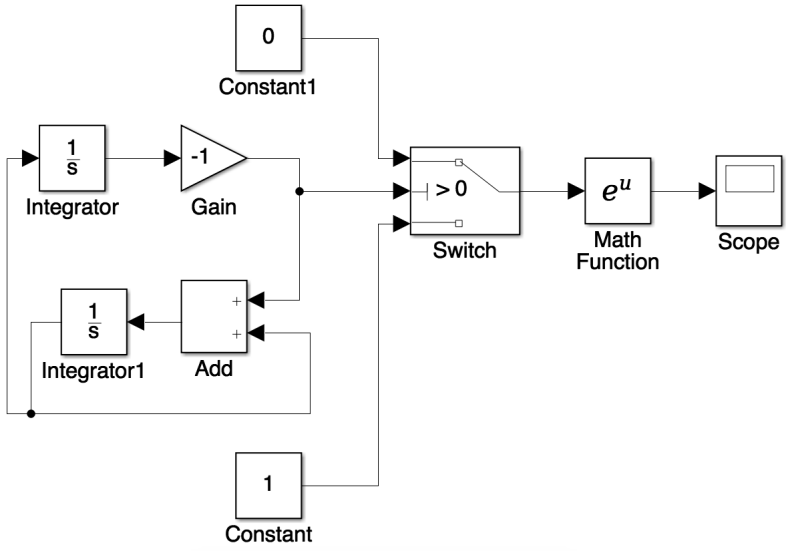

FIGURE 7.4: Simple Example of a hybrid model with a Switch block

We call the quadruple

$$(\mathfrak{P}, \mathfrak{E}, \mathfrak{I}, \mathfrak{O})$$

*the **enriched partitioning** of the Data Flow Model.*

The idea of adding the interface vertices to each partition is that we obtain complete Simulink submodels if we consider only the subgraph induced by a partition. It comprises output and input vertices, the *interface* vertices. This allows us to formulate the equivalence checking and epsilon propagation approach partition-wise.

Example 3. *Figure 7.4 shows an example for a simple hybrid model. Figure 7.5, which is an output of our implementation, shows the respective enriched partitioning for one data flow model (where the first data port is active) as provided by our implementation. The dark areas are the partitions. The Simulink model blocks are represented as vertices in the graph. Edges are depicted as arrows. The dotted arrows are references to the interfaces of the subsequent partition. The figure shows the added interface vertices in two partitions. In our implementation, we also have added a data input vertex to the block at the inactive data port.*

With the enriched partitioning, we close the syntactical preliminaries and proceed with the equivalence check. In the next section, we describe the subsequent process step, the symbolic equivalence check.

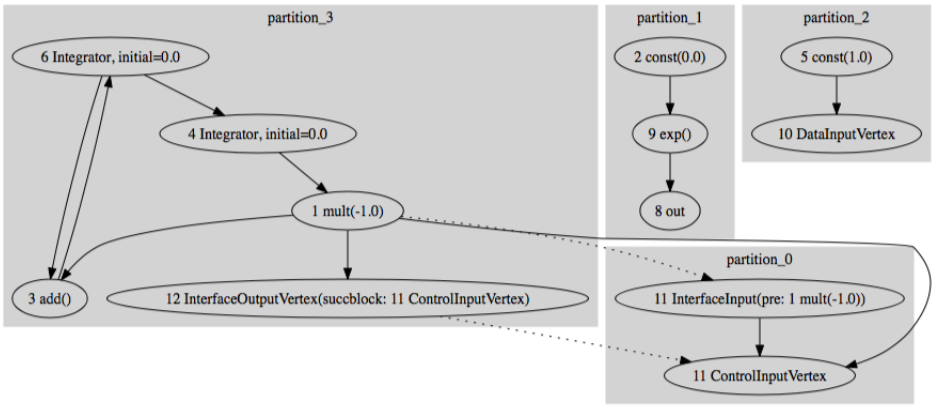

FIGURE 7.5: Enriched Partitioning for the example

7.3 Symbolic Equivalence Checking

After the calculation of the data flow models and the partitioning, we proceed with the symbolic equivalence check. The goal of this step is to establish equality of the ARs between pairs of partitions of the enriched partitioning in pairs of data flow models between source and target model. If this succeeds, we are able to apply Theorems 1 and 2 because each partition is either a purely time-discrete submodel, or a purely time-continuous submodel.

We describe the symbolic equivalence checking process in two steps. Firstly, we describe how we establish symbolic equivalence between a pair of partitions, one from an enriched partitioning of a data flow model of the source and one from the target model. Secondly, we describe how we use this partition-wise check to establish symbolic equivalence of the overall model.

7.3.1 Partition-Wise Symbolic Equivalence Checking

Consider we have two partitions P_1, P_2 originating from an enriched partitioning of a data flow model of the source model and an enriched partitioning of a data flow model from the target model. To check for symbolic equivalence between both partitions, we proceed as follows.

1. We extract the ARs from the partitions yielding equations for the output vertices and (if present) the stateful vertices. Note that in our enriched partitioning, each partition has at least an output vertex - either an output vertex originating from an output block of the model, or an interface output vertex expressing a point of observation for subsequent partitions. Hence, in any case, we obtain at least an output equation for a partition.

2. If stateful vertices are present, we extract the stateful equations directly in Laplace transformed or z-transformed form.

3. We relay the extracted terms for the check of symbolic equivalence to a Computer Algebra System (CAS).

In the first step, we extract our AR from each partition. If stateful blocks are present in a partition, we have a linear ODE or difference equation system per assumption. We express these equation systems in the second step directly by their Laplace or z-transformed [EA06, Kai80]. Please refer to Sections 2.2.4 and 2.2.4 for background information regarding Laplace and z-transform. Moreover, we derive an equation describing the output for each partition.

We agree on the notation of the following variables.

- y_i to the output of each stateful block, e.g. Integrator, Unit Delay, Transfer Function block.
- i_j the interface input vertices, i.e., elements from I_{P_i}.
- o_i analogously for the interface output vertices, i.e., elements from O_{P_i}.

For each partition, we extract an output equation describing the observation at its exits:

$$\mathbf{o} = f(\mathbf{i}, \mathbf{y})$$

We denote the equations in vectorial form in bold font. The dimension is determined by the amount of occurring output variables in the partition. For non-stateful partitions that have been defined as solutions by the user in the user-defined partitioning step, we additionally apply Laplace transform or z-transform on the expression via the CAS. If stateful blocks are present in a partition, we extract the equation directly in Laplace- or z-transformed format. We illustrate this for the time-continuous case and the associated Laplace transform. The time-discrete case is applied with the z-transform analogously.

We extract the Laplace transformed state equation:

$$\boldsymbol{Y} = A(s)\boldsymbol{Y} + \boldsymbol{I}(s)$$

We denote Laplace transformed variables with capital letters, i.e., $Y := \mathcal{L}(y)$ where $y : \mathcal{I} \to \mathbb{R}$ is a function fulfilling the condition for existence of the Laplace transformed as described in Section 2.2.4. This means, y is of at most exponential growth (cf. Lemma 8). The matrix $A(s)$ comprises expressions depending on the complex-valued variable s, which are rational polynomials. These polynomials are determined by the stateful block: For instance, Integrator blocks yield the expression $\frac{1}{s}$,

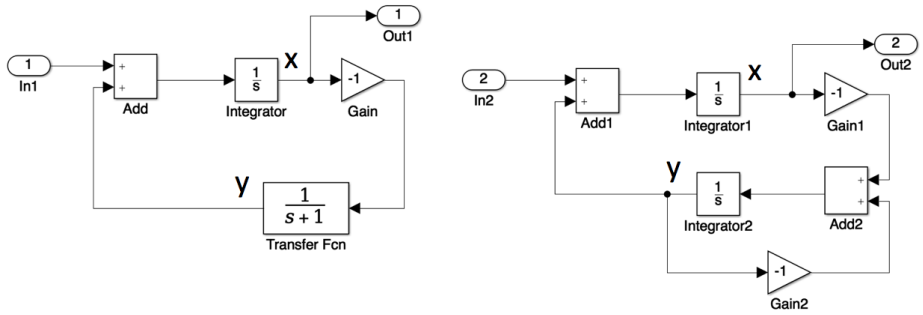

FIGURE 7.6: Laplace Transformed AR Extraction Example

Transfer Function blocks yield a polynomial $\frac{p(s)}{q(s)}$ given as parameter. The vector $\boldsymbol{I}(s)$ contains incoming signal variables and constants resulting from initial values of stateful blocks, multiplied with expressions depending on s. This equation is transformed to

$$\boldsymbol{Y} = (Id - A(s))^{-1}\boldsymbol{I}(s).$$

In the formula, Id denotes the identity matrix. This operation, in particular the calculation of the matrix inverse, is possible because the rank of $Id - A$ is always equal to the dimension of A.

Example 4. *Figure 7.6 illustrates an example for a time-continuous partition. There, the interfaces to preceding and subsequent partitions are represented by an input port and an output port. We have a source and a target model in the figure. The stateful blocks are associated with variables x, y. We obtain $o = x$ as output equations in both cases, and*

$$\binom{X}{Y} = \begin{pmatrix} 1 & -\frac{1}{s} \\ \frac{1}{s+1} & 1 \end{pmatrix}^{-1} \begin{pmatrix} \frac{1}{s}I \\ 0 \end{pmatrix} \binom{X}{Y} = \begin{pmatrix} 1 & -\frac{1}{s} \\ \frac{1}{s} & 1+\frac{1}{s} \end{pmatrix}^{-1} \begin{pmatrix} \frac{1}{s}I \\ 0 \end{pmatrix}$$

as state equations for the respective models.

After we have extracted the equations, the symbolic verification of the equivalence between two partitions is performed with the help of the extracted equations via a Computer Algebra System (CAS).

For partitions without stateful blocks, we have only two output equations, one from the source and one from the target model, i.e., $\boldsymbol{o}_1 = f_1(\boldsymbol{i}), \boldsymbol{o}_2 = f_2(\boldsymbol{i})$. We simply ask the CAS to symbolically unify the terms

$$f_1(\boldsymbol{i}) = f_2(\boldsymbol{i})$$

For partitions with stateful blocks, we have output equations

$$\boldsymbol{o}_1 = f_1(\boldsymbol{y}, \boldsymbol{i}), \boldsymbol{o}_2 = f_2(\boldsymbol{y}, \boldsymbol{i})$$

and state equations

$$\boldsymbol{Y}_1 = (Id - A_1(s))^{-1}\boldsymbol{I}(s), \boldsymbol{Y}_2 = (Id - A_2(s))^{-1}\boldsymbol{I}(s)$$

We ask the CAS to symbolically verify the equivalence of

$$f_1(\boldsymbol{y}, \boldsymbol{i}) = f_2(\boldsymbol{y}, \boldsymbol{i})$$

where the variables $\boldsymbol{y}$ have been replaced by the expressions in the resulting vectors of the state equations.

The case of partitions where one represents an ODE or difference equation, the other represents the analytical solution, which we obtain in the user-defined partitioning step, reduced to the case where we have two stateful equations because we have also transformed the analytical solution with Laplace transform or z-transform. The only difference is that in these cases, no input variables occur in the state equations. Instead, in both expressions are only polynomial that depend on s or z. These expressions are analogously checked for symbolic equality.

Example 5. *In our example 4 from Figure 7.6, we ask for instance if*

$$\pi_1\left(\begin{pmatrix} 1 & -\frac{1}{s} \\ -\frac{1}{s+1} & 1 \end{pmatrix}^{-1} \begin{pmatrix} \frac{1}{s}I \\ 0 \end{pmatrix}\right) = \pi_1\left(\begin{pmatrix} 1 & -\frac{1}{s} \\ \frac{1}{s} & 1+\frac{1}{s} \end{pmatrix}^{-1} \begin{pmatrix} \frac{1}{s}I \\ 0 \end{pmatrix}\right)$$

where π_1 is the projection on the first component of the vector. We take π_1 because the term describing the output is just x, i.e., the first component of the vector.

Equivalence checking of two time-discrete partitions works analogously to the time-continuous case if the linear ODE system is replaced by a linear difference equation system and the Laplace transform by the z-transform.

In the next paragraph, we describe how this partition-wise equivalence check is utilised to obtain an equivalence check for the overall model.

7.3.2 Model Equivalence Checking

For each data flow model with the same configuration in source and target model, we perform symbolic equivalence checking with the help of a CAS as described in the previous subsection. If this is successful for all data flow models, the overall

symbolic equivalence check is successful. Within data flow models, we perform the symbolic equivalence check partition-wise by moving along the partial order imposed by the partitionings as described in the previous paragraph. Note that this partition-wise check is semi-automated only: Consider two partitionings. We move along the partial order, i.e., we start at minimal elements and perform the partition-wise equivalence check. Once verified, we move on with the minimal elements succeeding the partitions we just successfully verified for symbolic equivalence. If the minimal element at the current step is the smallest element, i.e., if there is only one minimal element, it is obvious which partitions need to be checked for equivalence because there is only one choice. However, if there is more than one minimal element, we check each possible pair, and give a feedback to the user which check has been successful. The user is then asked to confirm that the correct pair has been selected. This is the second manual step in our automation approach.

If symbolic equivalence has been established for all partitions, the partitioning and consequently the respective data flow model has been successfully verified for symbolic equivalence. This in turn allows us to apply Theorems 4 and 5 respectively for all partitions, which in turn guarantees approximately equivalent behaviour for the models in the partitions.

Our symbolic equivalence check is sound because the partitioning imposes a partial order. If the user guarantees equivalence of the ingoing signals of the overall model, our partition-wise symbolic equivalence check can assume the equality of ingoing signals in a partition, i.e., at the interface vertices. Hence, we follow an assume-guarantee strategy.

The next step after establishing the symbolic equivalence for all Data Flow Models partition-wise is to calculate the local precision ε_l if necessary, and to propagate this throughout the partitions.

7.4 Epsilon Calculation and Propagation

In this section, we explain how, within a data flow model M_χ, 1) the precision of the approximate bisimulation is calculated locally, i.e., for stateful time-continuous parts within a partition where the refactoring replaces an ODE by its analytical solution yielding $\varepsilon_l(\#\mathcal{I})$ ($\mathcal{I}$ the simulation interval), and 2) how this local $\varepsilon_l(\#\mathcal{I})$ is propagated through the subsequent partitions yielding a global $\varepsilon_g(\#\mathcal{I})$ for the data flow model M_χ. The latter fulfills two purposes: 1) to obtain a global precision $\varepsilon_g(\#\mathcal{I})$ for the approximate bisimulation of the overall data flow model M_χ, and 2) to be able to check for possible time delays at control flow elements. We have performed the calculations described in this section with Mathematica [Wol15].

The key idea of our epsilon calculation and propagation approach is the following: We have purely time-discrete or purely time-continuous submodels in each partition p of our partitioning. For these submodels, we can apply Theorems 1 or 5. This yields $\varepsilon_{l_p}(\#\mathcal{I})$ at the output of our enriched partitioning. This value $\varepsilon_{l_p}(\#\mathcal{I})$ in turn is transferred to succeeding partitions. If $\varepsilon_{l_p}(\#\mathcal{I}) > 0$, this implies that the input signals of a directly succeeding partition p' are disturbed by $\varepsilon_{l_p}(\#\mathcal{I})$. For partition p' we can again apply Theorems 1 or 5. However, these theorems implicitly assume an undisturbed signal. We therefore estimate an additional error due to the disturbance of the input signals of magnitude $\varepsilon_{l_p}(\#\mathcal{I})$. This additional disturbance consists of a conditional error (the mathematical problem is disturbed), and a convergence error (the effect on the numerical algorithm due to the disturbance). Please refer to Section 2.2.5 in Background Chapter 2 for details to numerical errors. Since we have limited our subset of supported Simulink models for this automation approach to those that can be represented by linear ODEs or difference equations, we can always provide such an estimate. Our approach therefore is capable of dealing with alternations between time-discreteness and time-continuity between partitions.

Calculation of Local Epsilon

To keep the notation simple, and to avoid too many subindices, we assume that the calculation refers to a data flow model M_χ, and a partition p, without explicitly mentioning this in the formulas. Hence, with ε_l, we mean the precision of the approximate bisimulation at the exits of the partition p in the data flow model M_χ.

We start with the calculation of the *local* ε_{l_y} at the exit signals of stateful blocks of a partition p (y indicates that we refer to the state equations). For all time-discrete partitions, it is 0 - independent from the fact whether a refactoring took place in the partition or not. Furthermore, it is 0 for time-continuous systems if it is (partially) replaced with a Transfer Function block, or an arithmetic change has taken effect, but the ODEs remain unchanged. The reason for that is that provided both models are simulated with the same step size and the same numerical method, the calculations yield the same values.

The precision ε_{l_y} can only be greater than 0 for 1) a time-continuous partition and 2) if the ODE is resolved, i.e., the refactoring replaces the ODE $\frac{d}{dt}\mathbf{y}(t) = A\mathbf{y}(t) + \mathbf{i}(t), \mathbf{y}(0) = \mathbf{y}_0$ by its solution λ or the other way around. Otherwise, the ODE is not resolved, but the partitions are symbolically equivalent, which can only be the case if an arithmetic refactoring has been performed, and the ODEs are the same. As we know from Chapter 6 this implies $\varepsilon_{l_y} = 0$ because if the ODEs are the same, the approximation technique in both models yields the same values. The matrix A,

which we use for the epsilon calculation, can be extracted analogously as described in the previous subsection.

This also implies that refactorings causing a local $\varepsilon_l > 0$ are only possible at minimal elements of the partitioning. Examples for such refactorings are the alternative expression of components that are desired, but not implementable due to engineering constraints. An example would be the replacement of a Sine Wave Generator block by the ODE $\frac{d^2}{dt^2}y = -y, y(0) = 0, \frac{d}{dt}y(0) = 1$.

As described in Section 6.3.1, the calculation of ε_{l_y} in this case is generally determined by a formula

$$\varepsilon_{l_y}(\#\mathcal{I}) = f_{err}\left(L, h, \#\mathcal{I}, \max_{t\in\mathcal{I}}\left\|\frac{d^2}{dt^2}\lambda(t)\right\|_\infty\right)$$

where L is the Lipschitz constant, h the simulation step size, $\#\mathcal{I}$ the length of the simulation interval, and λ is the unique solution of the ODE. In this section, we have also provided techniques to calculate the parameters of the formula. In particular,the function f_{err} depends on the selected simulation technique, h and $\#\mathcal{I}$ are given as parameters by the user. We obtain an estimate for the Lipschitz constant L by applying Lemma 7. Since the matrix A has constant coefficients, we directly obtain an estimate for L. The Lemma says, that the estimate can be calculated by $\|\mathcal{J}_f(\boldsymbol{y})\|_\infty$ where $f(t) = A\boldsymbol{y}(t) + \boldsymbol{i}(t)$. For $A = (a_{i,j})_{1\le i,j\le n}$, this leads to

$$\|\mathcal{J}_f(\boldsymbol{y})\|_\infty = \|A\|_\infty = \max_{1\le i\le n}\sum_{j=1}^{n}|a_{ij}|$$

To calculate $\max_{t\in\mathcal{I}}\left\|\frac{d^2}{dt^2}\lambda(t)\right\|_\infty$, we use the available solution λ. The CAS can calculate the maximum in the simulation interval automatically.

The process yields a disturbance $\varepsilon_{l_y}(\#\mathcal{I})$ associated with the exits of the stateful blocks. To calculate the $\varepsilon_l(\#\mathcal{I})$ for partition p, we use the term of the output equation of the partition $f_o(\boldsymbol{y}, \boldsymbol{i})$. Depending on the function f_o, the disturbance $\varepsilon_{l_y}(\#\mathcal{I})$ can increase. For instance for sums we obtain $\varepsilon_{l_{y_1}}(\#\mathcal{I}) + \varepsilon_{l_{y_2}}(\#\mathcal{I})$. For Gain blocks, i.e., multiplication with constant λ, we obtain $\lambda\varepsilon_{l_y}(\#\mathcal{I})$ as resulting perturbance and so on.

7.4.1 Epsilon Propagation through Partitions

The disturbance ε_l has an impact on subsequent partitions. If stateful blocks are present, this means in particular the following: Consider one input to a partition p' subsequent to our partition p with ε_l. Then, this partition p' is disturbed by ε_l at the interface vertices. To illustrate this, consider Figure 7.7. On the left hand side,

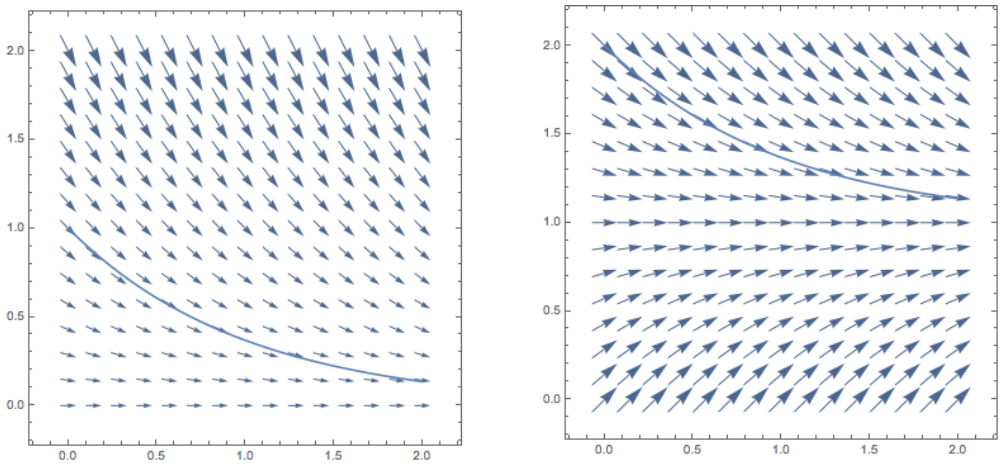

FIGURE 7.7: Vectorfields for Disturbed and Undisturbed Solution

an undisturbed vectorfield induced by an ODE $\frac{d}{dt}y(t) = f(t, y(t))$ together with one solution is depicted, on the right hand side the disturbed counterpart is depicted. The arrows in the vectorfields depict the inclination of the solution for each point (t, y) determined by the ODE, i.e., at point (t, y), the vector points in the direction $(1, f(t, y))$. As illustrated there, the disturbance causes the initial value to shift and the inclination to change. Hence, we obtain a different solution. We provide an estimate for the maximal distance between these solutions. This is called *conditional error* $\varepsilon_{y_{cond}}(\#\mathcal{I})$ in accordance with the numerical errors as introduced in Section 2.2.5.

For time-discrete partitions, this conditional error $\varepsilon_{y_{cond}}(\#\mathcal{I})$ is the final resulting error. For time-continuous partitions, the disturbance $\varepsilon_{y_{cond}}(\#\mathcal{I})$ in turn yields an additional approximation error, a *convergence error* $\varepsilon_{y_{conv}}(\#\mathcal{I})$, which can in turn be calculated via the error formula f_{err} (cf. Section 2.2.5). The overall disturbance at the Interface Output vertices depend on the output equations analogously as in the calculation of the local epsilon.

Conditional Error Calculation

We firstly calculate an estimate of the disturbance caused by the ODE or difference equation. For non-stateful partitions, it works as for the output equations to calculate $\varepsilon_l(\#\mathcal{I})$. At stateful partitions, we describe the time-continuous case. The time-discrete case works analogously. In the case of stateful blocks, we have ODEs

$$\frac{d}{dt}\mathbf{y}(t) = A\mathbf{y}(t) + \boldsymbol{i}(t), \boldsymbol{y}(0) = \boldsymbol{y}_0, \frac{d}{dt}\hat{\boldsymbol{y}}(t) = A\hat{\boldsymbol{y}}(t) + \boldsymbol{i}(t) + \varepsilon_l(\#\mathcal{I}), \hat{\boldsymbol{y}}(0) = \boldsymbol{y}_0 + \varepsilon_l(\#\mathcal{I})$$

where the latter is the disturbed ODE and the disturbance $\varepsilon_l(\#\mathcal{I})$ results from preceding partitions.

We now show in the following Lemma how we can obtain an estimate of the maximal deviation between undisturbed and disturbed ODE.

Lemma 19 (Maximal Disturbance of Disturbed and Undisturbed ODE)**.** *Let*

$$\frac{d}{dt}\mathbf{y}(t) = A\mathbf{y}(t) + \boldsymbol{i}(t), \boldsymbol{y}(0) = \boldsymbol{y}_0, \frac{d}{dt}\hat{\boldsymbol{y}}(t) = A\hat{\boldsymbol{y}}(t) + \boldsymbol{i}(t) + \varepsilon_l, \hat{\boldsymbol{y}}(0) = \boldsymbol{y}_0 + \varepsilon_0$$

be undisturbed and disturbed ODE with constants $\varepsilon_l, \varepsilon_0$. Let the solutions of the ODEs be $\boldsymbol{\lambda}$ and $\hat{\boldsymbol{\lambda}}$. Then

$$\hat{\boldsymbol{\lambda}}(t) = \boldsymbol{\lambda}(t) + \tilde{\boldsymbol{\lambda}}(t)$$

where $\tilde{\lambda}(t)$ is the solution of the ODE

$$\frac{d}{dt}\tilde{\boldsymbol{y}}(t) = A\tilde{\boldsymbol{y}}(t) + \varepsilon_l, \tilde{\boldsymbol{y}}(0) = \varepsilon_0$$

Hence, the maximal disturbance can be estimated by

$$\|\boldsymbol{\lambda} - \hat{\boldsymbol{\lambda}}\|_\infty = \max_{t\in\mathcal{I}} \|\tilde{\boldsymbol{\lambda}}(t)\|_\infty$$

Proof. We show that $\hat{\boldsymbol{\lambda}} = \boldsymbol{\lambda} + \tilde{\boldsymbol{\lambda}}$ solves the initial value problem. Firstly, we show that it fulfills the ODE. This follows from

$$\frac{d}{dt}\hat{\boldsymbol{\lambda}}(t) = \frac{d}{dt}(\boldsymbol{\lambda} + \tilde{\boldsymbol{\lambda}})(t) = A\boldsymbol{\lambda}(t) + \boldsymbol{i}(t) + A\tilde{\boldsymbol{\lambda}}(t) + \varepsilon_l$$

$$= A(\boldsymbol{\lambda} + \tilde{\boldsymbol{\lambda}})(t) + \boldsymbol{i}(t) + \varepsilon_l = A\hat{\boldsymbol{\lambda}}(t) + \boldsymbol{i}(t) + \varepsilon_l$$

Secondly, we verify if the intial value equation is fulfilled.

$$\tilde{\boldsymbol{\lambda}}(0) = (\boldsymbol{\lambda} + \hat{\boldsymbol{\lambda}})(0) = y_0 + \varepsilon_0$$

Since for linear ODEs, the solutions are unique because the right hand side of the equation is Lipschitz-continuous, this concludes the proof. □

We use this lemma to obtain disturbance error by relaying the ODE

$$\frac{d}{dt}\tilde{\boldsymbol{y}}(t) = A\tilde{\boldsymbol{y}}(t) + \varepsilon_l(\#\mathcal{I}), \tilde{\boldsymbol{y}}(0) = \varepsilon_l(\#\mathcal{I})$$

to the Computer Algebra System and asking for a solution, which always exists for linear ODEs. The CAS also tells us the maximum in the simulation interval $\mathcal{I}$. This yields the disturbance $\varepsilon_{y_{cond}}(\#\mathcal{I})$.

The time-discrete case with difference equations can analogously be applied. We demonstrate this by providing an analogous Lemma for difference equations.

Lemma 20 (Maximal Disturbance of Disturbed and Undisturbed Difference Equation)**.** *Let*

$$\mathbf{y}_{n+1} = A\mathbf{y}_n + \boldsymbol{i}_n, \boldsymbol{y}_0 \text{ initially given }, \hat{\boldsymbol{y}}_{n+1} = A\hat{\boldsymbol{y}}_n + \boldsymbol{i}_n + \varepsilon_l, \boldsymbol{y}_0 + \varepsilon_0 \text{ initial value}$$

be undisturbed and disturbed difference equation with constants $\varepsilon_l, \varepsilon_0$. Let the solutions of the difference equations be $(\boldsymbol{\lambda}_n)_n$ and $(\hat{\boldsymbol{\lambda}})_n$. Then

$$\forall n \in \mathbb{N} : \hat{\boldsymbol{\lambda}}_n = \boldsymbol{\lambda}_n + \tilde{\boldsymbol{\lambda}}_n$$

where $(\tilde{\lambda}_n)_n$ is the solution of the difference equation

$$\tilde{\boldsymbol{y}}_{n+1} = A\tilde{\boldsymbol{y}}_n + \varepsilon_l, \varepsilon_0 \text{ the initial value}$$

Hence, the maximal disturbance can be estimated by

$$\|\boldsymbol{\lambda}_n - \hat{\boldsymbol{\lambda}}_n\|_\infty = \max_{n\in\mathbb{N}} \|\tilde{\boldsymbol{\lambda}}_n\|_\infty$$

Proof. We have to show that $\hat{\boldsymbol{\lambda}}_n = \boldsymbol{\lambda}_n + \tilde{\boldsymbol{\lambda}}_n$ solves the difference equation. We perform this via natural induction. For $n = 0$, we have

$$\hat{\boldsymbol{\lambda}}_0 = \boldsymbol{y}_0 + \varepsilon_0 = \boldsymbol{\lambda}_0 + \tilde{\boldsymbol{\lambda}}_0$$

Consider now

$$\hat{\boldsymbol{\lambda}}_{n+1} = A\hat{\boldsymbol{\lambda}}_n + \boldsymbol{i}_n + \varepsilon_l = A(\boldsymbol{\lambda}_n + \tilde{\boldsymbol{\lambda}}_n) + \boldsymbol{i}_n + \varepsilon_l$$

The last equation holds due to the inductive assumption. We continue as follows

$$= A\boldsymbol{\lambda}_n + \boldsymbol{i}_n + A\tilde{\boldsymbol{\lambda}}_n + \varepsilon_l = \boldsymbol{\lambda}_{n+1} + \tilde{\boldsymbol{\lambda}}_{n+1}$$

This concludes the proof. □

Convergence Error Calculation

The change from an undisturbed to disturbed solution causes a change of the convergence error, i.e., the error due to the approximation of the ODEs. This applies only for time-continuous partitions since there is no approximation at time-discrete partitions. As described in the local epsilon calculation step, the convergence error, i.e., the sum of consistency and stability error for single step approximation

techniques can be estimated by a function

$$\varepsilon_{l_y}(\#\mathcal{I}) = f_{err}\left(L, h, \#\mathcal{I}, \max_{t\in\mathcal{I}}\left\|\frac{d^2}{dt^2}\lambda(t)\right\|_\infty\right)$$

We know from Lemma 19 that we can represent the solution of the disturbed ODE $\hat{\boldsymbol{\lambda}} = \boldsymbol{\lambda} + \tilde{\boldsymbol{\lambda}}$. Moreover, we know that the convergence error of the exact solution $\boldsymbol{\lambda}$ is in both models the same. The reason for this is that at partitions that are not minimal elements in the partitioning, no replacement of ODE by analytical solution or vice versa may have occurred. Hence, the ODEs in such partitions remain unchanged and therefore as pointed out in Section 6.3.1, the convergence error of the undisturbed solution is 0. Therefore, we can conclude that the convergence error we need to estimate can be calculated by

$$\varepsilon_{y_{conv}}(\#\mathcal{I}) = f_{err}\left(L, h, \#\mathcal{I}, \max_{t\in\mathcal{I}}\left\|\frac{d^2}{dt^2}\tilde{\boldsymbol{\lambda}}(t)\right\|_\infty\right)$$

We have calculated $\tilde{\boldsymbol{\lambda}}$ in the previous step and therefore are able to calculate the estimate. Overall, this sums up to

$$\varepsilon_{l_y}(\#\mathcal{I}) = \varepsilon_{y_{cond}}(\#\mathcal{I}) + \varepsilon_{y_{conv}}(\#\mathcal{I}).$$

As described in the calculation of the local epsilon, we use this $\varepsilon_{l_y}(\#\mathcal{I})$ to obtain the estimate $\varepsilon_l(\#\mathcal{I})$ for the partition, i.e., the disturbance at the interface output vertices. We perform the calculation and propagation of the $\varepsilon(\#\mathcal{I})$ as described in the previous subsections for all data flow models and take the maximum to obtain the overall precision of the approximate bisimulation.

7.5 Approximate Bisimulation Verification

Finally, we describe how we combine the steps of our automation approach to be able to apply Theorem 6 to conclude approximate bisimulation of two given models with precision ε.

As described in Section 7.2, we calculate all possible data flow models, i.e., models describing the data flow for all possible configurations. Moreover, we perform a partitioning. This results in a partitioning graph determined by the underlying graph structure of the original model. We combine cycles and non-cyclic parts. Each of the partitions of the partitioning can be considered as fully executable submodels. We perform the symbolic equivalence check for all data flow models with the same configuration in source and target model, i.e., under

the same switching condition. We perform the check following the graph structure on partitioning level. The user is asked to guarantee that our check is compliant with the structure of the partitionings, i.e., that the proper partitions within the partitionings have been checked for symbolic equivalence. We apply an assume-guarantee strategy: Equivalences of output signals of partitions imply equivalence of respective ingoing signals of subsequent partitions. Overall, this yields a sound symbolic equivalence check. Together with a check if all conditions at switch blocks in source and target model are the same, this yields an overall applicability of Theorem 6. The calculation of the precision $\varepsilon(\#\mathcal{I})$ of the approximate bisimulation of source and target model is performed firstly locally at partitions that are minimal elements in the partitioning graph, and secondly by a sound propagation through the graph structure of the partitioning. For each data flow model M_χ, we perform this calculation of the precision of the approximate bisimulation yielding for each χ values $\varepsilon_{\chi_g}(\#\mathcal{I})$. In accordance with Theorem 6, we can conclude that

$$\varepsilon(\#\mathcal{I}) := \max_{\chi \in X} \varepsilon_{\chi_g}(\#\mathcal{I})$$

is the precision of approximate bisimulation of the overall model. Theorem 6 only applies for models without time-hybrid components. However, since our ε provides an overapproximation for our subset of time-hybrid models, the theorem's statement directly transfers to the case of our automation approach.

Checks for Time-Delayed Switching Behaviour

It remains to discuss the additional criteria for a potential time-delayed behaviour as discussed in Subsection 6.4.2. There, we have provided criteria for models that are not time-hybrid. We have on the one hand a sound overapproximation of ε, but on the other hand may have time-hybrid submodels. We therefore re-consider the conditions as described there and apply them in the following way. Consider an enriched partitioning of a data flow model as described in this chapter. Consider furthermore a control output block that has been created in the data flow model calulation step (and intuitively keeps the control signal) in a partition p of the partitioning. Then, we distinguish the following cases (for more than one control output block in the same partition, the cases apply simultaneously for all blocks):

1. If the model in p is purely time-continuous, we apply exactly the same conditions as described in Section 6.4.2. There, a solution of the trajectory for the control signal is required. We simply need to calculate the solution within the partition (and not the solution of the overall model for the control signal)

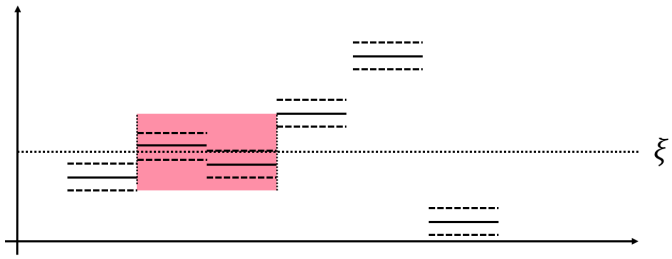

FIGURE 7.8: Intervals for potential time-delayed switching behaviour for time-discrete disturbed partitions

because the ε_c at the control signal already considers disturbances from preceding partitions. Since we have linear ODEs with constant coefficients only, these are always solvable.

2. For time-discrete models at control signals, conditions from Section 6.4.2 say, that we have nothing to check since $\varepsilon_c = 0$ in these cases. However, this is only true if the overall model is time-discrete. However, we may have a purely time-continuous partition preceding our time-discrete p implying a disturbance $\varepsilon_c > 0$, which we calculate and propagate as described in Subsection 7.4 of this chapter. Hence, we need to re-consider the conditions for time-delayed switching behaviour in this case.

So, for the time-discrete, disturbed case, we re-consider the conditions for time-delayed switching behaviour. In Section 6.4.2, we have provided three conditions. Firstly, if the trajectory including ε_c is too far away from the threshold of the switch to cross it, no switch occurs in both models and therefore no time-delayed switching behaviour may occur. This criterion translates without change to our case. The second condition, which treats a threshold crossing between sample steps and therefore the threshold crossing is undetected, is not applicable because time-discrete elements keep the signal constant between sample steps. Hence, if a threshold crossing occurs, it definitely is detected. The third and last condition is used to detect intervals where time-delayed switching behaviour may occur. To this end, we have checked zero-crossings for the functions $\lambda(t) - \varepsilon$ and $\lambda(t) + \varepsilon$. Translated to the disturbed time-discrete case, this means that time-delayed switching behaviour may occur in intervals $[n_1, n_2]$ where

$$\forall n : n_1 \leq n \leq n_2 : \|a_n - \xi\|_\infty \leq \varepsilon_c,$$

where $(a_n)_n$ is the sequence fulfilling the AR at the control output block, ξ is the threshold at the switch block.

Figure 7.8 illustrates the condition. There, the trajectory of a control signal together with sourrounding ε environments is depicted. An interval where the condition as defined above is fulfilled and therefore time-delayed switching behaviour may occur, is marked. The other time steps do not fulfill the condition and therefore no time-delayed switching behaviour can happen there.

The checks for purely time-continuous partitions can be performed with the help of a CAS such as Mathematica as well. This is because such CAS are capable of calculating zero crossings of functions. For time-discrete disturbed partitions, a proof by natural induction or interval-wise check of the above condition is required. This needs to be performed either manually or with the help of a subsequent system. Hence, we have a restriction of our claim for full automation in this case. However, many cases, such as purely time-continuous control signals, or if $\varepsilon_c = 0$, are fully automatable supported.

7.6 Summary

In this chapter, we have described our automation approach. It is an application and extension of the methodology we have presented in Chapters 5 and 6. Our approach obtains two Simulink models and provides 1) an answer if the models are approximately equivalent, and 2) if positive, the precision ε of the approximate bisimulation. Our approach consists of the following steps. We start with the calculation of the data flow models for all configurations. They describe all possible data flows and are Simulink models without control flow elements. We proceed with a partitioning of the data flow models. The partitioning 1) separates the models into components, which can be verified for equivalence separately, and 2) into purely time-discrete and purely time-continuous parts. This way, we allow different types between partitions: time-discrete partitions can follow time-continuous partitions and vice versa. After the calculation of the data flow models and the partitioning, our approach proceeds with a symbolic equivalence check. We perform this for all data flow models, going through the partitions in the order of the directed graph imposed by the partitioning. Where the structure is not uniquely determined, the user is asked for confirmation that our symbolic verification was successful for the correct choice of pairs of partitions. Our equivalence check follows an assume-guarantee strategy. The partition-wise symbolic equivalence check extracts equations from the partitions, transforms them via Laplace transform or z-transform, and tries to establish symbolic unification via a Computer Algebra System. After symbolic equivalence has been established, we proceed with the calculation of the precision $\varepsilon(\#\mathcal{I})$ of the approximate bisimulation. To this end, we firstly calculate the local precision

$\varepsilon_l(\#\mathcal{I})$, located at the partition containing an ODE that has been replaced by its analytical solution by the refactoring. Then, we propagate this local disturbance $\varepsilon_l(\#\mathcal{I})$ through subsequent partitions to obtain a global precision $\varepsilon_g(\#\mathcal{I})$. We have modelled the epsilon propagation by extending the epsilon calculation of Theorems 1 and 2 to support models with disturbed ingoing signals. To exclude time-delayed behaviour at Switch blocks, we use the calculated precision ε_c at control output blocks to identify intervals where time-delayed switching behaviour may occur. We have updated the conditions from the last chapter, Subsection 6.4.2 that deal with undisturbed purely time-discrete or purely time-continuous to the case for disturbed submodels. With our automation approach, we have demonstrated an important application for automated verification of behavioural equivalence on data flow-oriented hybrid control modelling languages.

Chapter 8

Evaluation

We have implemented our approach and evaluated it by verifying the correctness of three refactorings in three case studies, namely a time-continuous model of an aircraft, a hybrid model of a water tank, and a hybrid model of a cruise control system. In this chapter, we describe important aspects of our implementation of our automation approach as described in Chapter 7. We introduce our case studies and provide experimental results. We proceed with a discussion of our approach. In particular, we discuss some limitations and opportunities of our approach. Moreover, we provide a discussion of the complexity of the involved computations of our implementation.

8.1 Implementation

Our implementation is based on a representation of Simulink in Java that has previously been developed in our research group Software and Embedded Systems Engineering in the course of the DFG research project Methods of Model Quality (MeMo) [HWS+11]. In MeMo, a parser has been developed that transforms a given Simulink model into an object representation in Java. This equips us with the means to access structural information of the model as well as attributes of the Simulink blocks like sample times via Java.

We have implemented our approach from Chapter 7 in Scala. Figure 8.1 provides an overview over the architecture of our implementation. The top part of the figure is the connection to the MeMo projectWe use the Simulink IR in Java from this project. Our implementation chiefly consists of two parts: the syntactical preparation engine, and the verification engine. The syntactical preparation engine performs three major tasks. Firstly, it calculates a graph-based representation of the Simulink model. In this process, the Simulink model is flattened, i.e., the subsystem hierarchy is dissolved and the subsystems are removed. Secondly, the data flow models for all configurations are calculated yielding models without control flow elements.

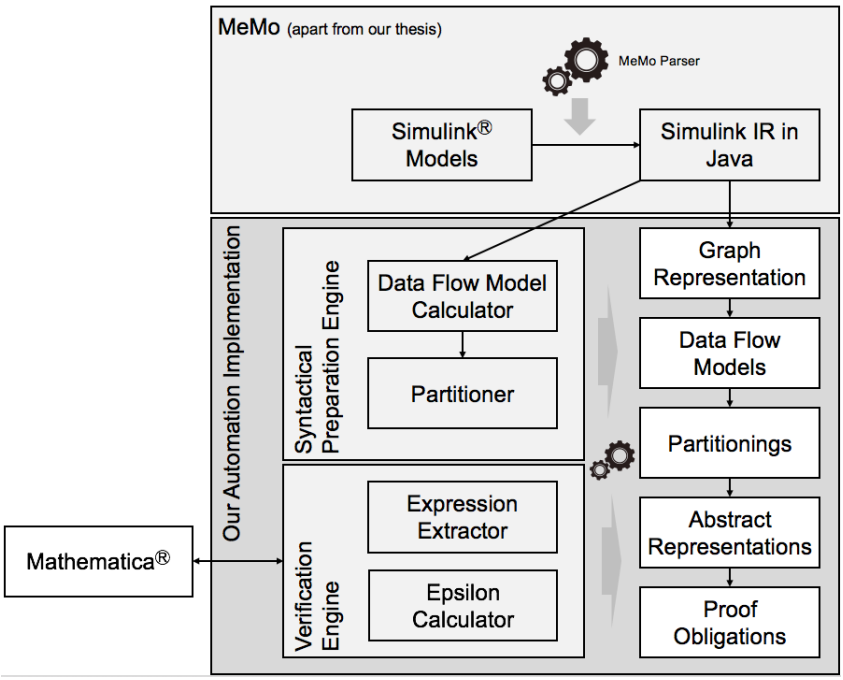

FIGURE 8.1: Implementation Architecture

Note that our current implementation only supports Switch blocks as control flow elements. Thirdly, we calculate the partitionings and the enriched partitionings as described in Chapter 7. In the partitioning, the user is asked to define partitions representing an ODE in the source model and its analytical solution in the target model, if applicable.

After the syntactical preparations, the verification engine proceeds. Firstly, the ARs are extracted. As discussed in Chapter 7, we directly extract the Laplace or z-transformed terms. In the Expression Extractor step, we also calculate matrix representations for each partition

$$\boldsymbol{Y}_1 = (Id - A_1(s))^{-1}\boldsymbol{I}(s), \boldsymbol{Y}_2 = (Id - A_2(s))^{-1}\boldsymbol{I}(s)$$

and calculate the proof obligations

$$f_1(\boldsymbol{y}, \boldsymbol{i}) = f_2(\boldsymbol{y}, \boldsymbol{i})$$

The symbolic equivalence checking is performed via the CAS Mathematica. Our verification engine coordinates these checks. Data flow models of source and target model with the same configuration are compared, the conditions at control flow elements are compared for equality. The checks in the data flow models follow an assume-guarantee strategy through the respective partitionings as described in

Chapter 7. If this check is successful, and if a replacement of an ODE by its analytical solution was part of the refactoring, which is identified in the partitioning step, then $\varepsilon > 0$. In this case, a calculation of the local ε_l at the ODE and analytical solution takes place. After that, the Epsilon Calculator propagates the local ε_l through the partitions to obtain the global ε_g as described in Chapter 7. We perform the symbolic verification for each pair of partition in the partitionings. The user needs to check if the correct pairs resulted in a successful verification, i.e., if for the correct pairs the result returned from the CAS was *true*.

There have been various Bachelor theses under our supervision that have contributed to the implementation. Most of them have been an interim step to obtain the final solution or have provided additional supporting tools. We provide a brief overview. In [Dan16], Armin Danziger has provided a visualisation of the Simulink IR. This is not part of the core implementation, but supports testing. In [Bas16], Johannes Basler has implemented an approach that enables verification for simple, closed models, i.e., models without input signals. This has been further improved to also cover open signals in [Sal17] by Ahmet Salman. Ahmet has implemented the equivalence check for open models, i.e., models with input signals using the CAS MATLAB Symbolic Math Toolbox. In the Bachelor thesis [Wor16], Johannes Wortmann has implemented the formal semantics [BC12]. This enables us to compare evaluations via the operational semantics with evaluations performed directly via Simulink.

8.2 Case Studies and Evaluation

In this section, we describe the case studies that we have used to evaluate our approach and present the experimental results.

8.2.1 Case Studies

To demonstrate the applicability of our approach, we have selected three case studies: 1) a time-continuous example from the Simulink examples website [The17], which is a model of the longitudinal flight control of the Grumman Aircraft F14, 2) a model for a water tank, and 3) a hybrid model for a cruise control system from [BC12].

F14 Model

The time-continuous F14 model [The17] controls the flight trajectory of an aircraft F14. The aircraft dynamics is influenced by the pilot's input signals and a wind gust model. The output is the flight trajectory evolving with time.

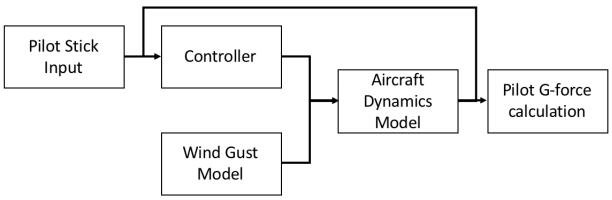

FIGURE 8.2: F14 Model Overview

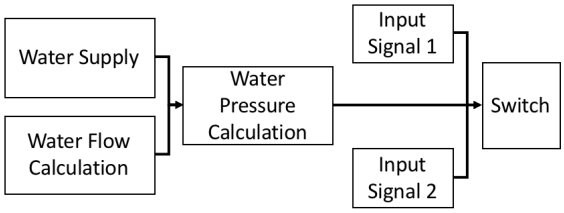

FIGURE 8.3: Water Tank Model Overview

Figure 8.2 depicts the top-level subsystem view of the F14 model. As we know from Chapter 7, only piecewise smoothness and at most exponential growth must be guaranteed for them.

Water Tank Model

The second case study models a water tank with changing water supply. It has been established in the course of David Skowronek's Bachelor thesis [Sko17]. Figure 8.3 provides a high-level overview. The model calculates the pressure in the tube. After reaching a certain threshold, the valve is opened. The valve's status is indicated by two input signals. A sinusoidal function models the periodic water fill over days. We have created this model on our own to provoke time-delayed behaviour at control flow elements.

Cruise Control Model

The hybrid cruise control model from [BC12] brings the vehicle to defined velocity. It contains time-discrete and time-continuous parts, and a Switch block. Figure 8.4 provides a high-level overview over the components of the model.

The input signals are used to define an initial velocity and the desired velocity. The time-continuous parts calculate the velocities and the response to reach the desired velocity in a defined period of time. The time-discrete part also calculates the response to reach the desired velocity. The output is the time-way diagram.

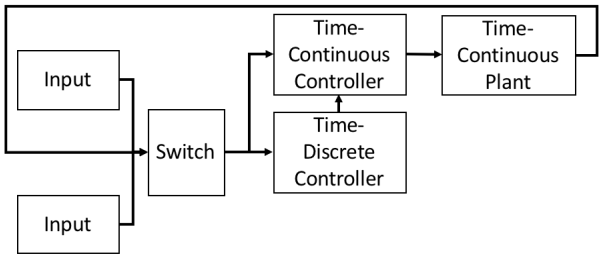

FIGURE 8.4: Cruise Control Model Overview

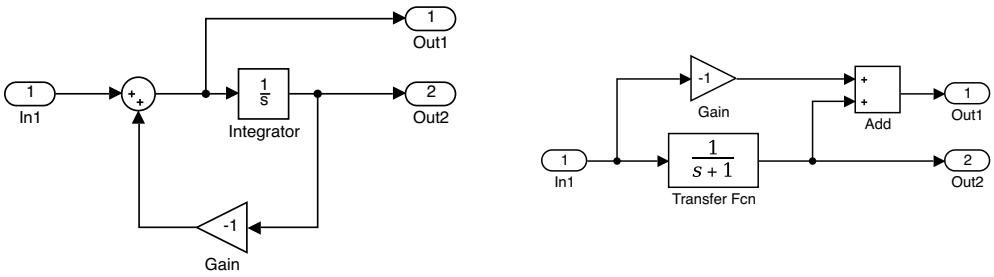

FIGURE 8.5: Replacement ODE by Transfer Function

Performed Refactorings

In each case study, we have performed several refactorings: we have replaced ODEs by Transfer Functions, performed arithmetic changes, and replaced an ODE being part of the water tank model by its analytical solution with an ε greater than 0.

Figure 8.5 provides an example refactoring performed in the F14 model. There, an ODE in the source model has been replaced by a Transfer Function in the target model. Note that the refactored model parts are contained within a feedback loop in the overall model.

Other examples of refactorings are arithmetic changes.

Figure 8.6 depicts an example where a gain, i.e., a multiplication with a constant is altered. The multiplicative constant is pulled into the Transfer Function. This yields a valid refactoring. Figure 8.7 shows another example. There, distributivity law is exploited. Examples for a replacement of an ODE by its analytical soution are

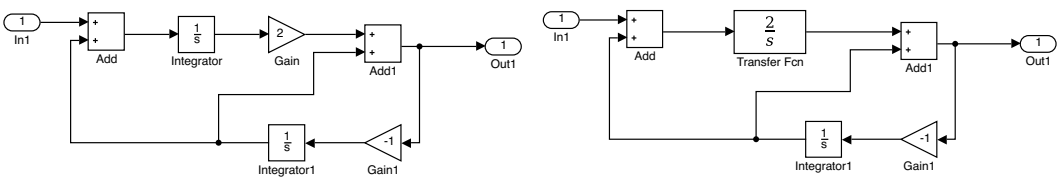

FIGURE 8.6: Arithmetic Refactoring Example 1

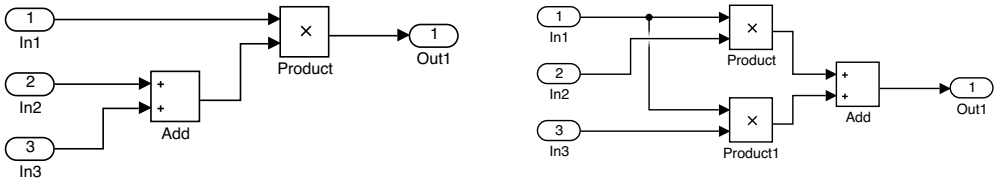

FIGURE 8.7: Arithmetic Refactoring Example 2

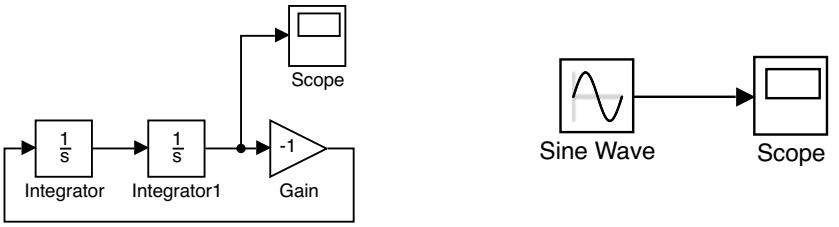

FIGURE 8.8: Solution of an ODE: Sine Wave

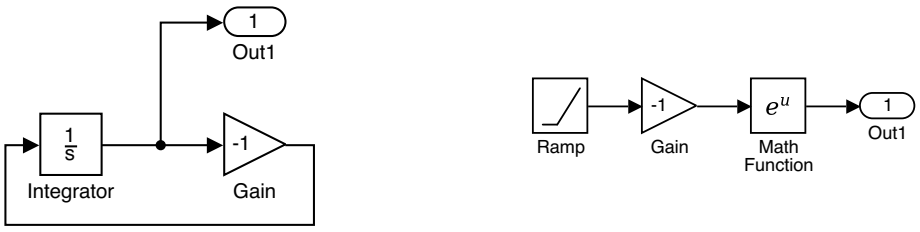

FIGURE 8.9: Solution of an ODE: Exponential Function

TABLE 8.1: Experimental Results Syntactical Preparations (CPU times in milliseconds)

Model	Blocks	Refactoring	Graph	DF Models	Total
Cruise Control	16	T2ODE	2	5	7
		Arithmetic	2	4	6
Water Tank	30	T2ODE	1	1	2
		Arithmetic	2	1	3
		ODE sol	1	1	2
F14	60	T2ODE	40	0	40
		Arithmetic	38	0	38

TABLE 8.2: Experimental Results Equivalence Checking (CPU times in milliseconds or format m:ss)

Model	Blocks	Refactoring	Partitioning	Check	Overall
Cruise Control	16	T2ODE	272	52316	52595
		Arithmetic	298	53174	53478
Water Tank	30	T2ODE	312	51512	51826
		Arithmetic	295	52016	52224
		ODE sol	298	51942	52242
F14	60	T2ODE	4:46	46560	5:33
		Arithmetic	4:45	46312	5:32

given in Figures 8.8 and 8.9. In Figure 8.8, the solution of the ODE is $\varphi(t) = \sin(t)$, in Figure 8.9 it is $\varphi(t) = e^{-t}$ (initial values are not depicted, but available as parameters). In our water tank model, we have replaced the ODE $\frac{d}{d}y(t) = 5000(1 + \sin(t)), y(0) = 5000$ by its analytical solution. We have experimented with various of these kinds of refactorings. We conducted valid and invalid changes. All were properly confirmed or denied. The precision ε has been calculated and propagated properly in all cases.

8.2.2 Experimental Results

Tables 8.1 and 8.2 provide an overview over the experimental results. In both tables, the case studies are in the first column together with the model size in the second column. The third column contains an abbreviation for the refactoring: T2ODE for Transfer Function to ODE (or vice versa), Arithmetic for arithmetic changes, and ODE sol for ODE solution, i.e., the replacement of an ODE by its analytical solution.

In each experiment, ε had the value 0, except for the water tank model. The replacement of the ODE in our water tank model by its analytical solution has yielded $\varepsilon(\#\mathcal{I} = 10) = 0.025$ at the control port for a simulation interval length of 10, sample step size 10^{-8} and Euler method as selected numerical technique. This $\varepsilon(\#\mathcal{I})$ increases linearly with the interval size in this case because the replaced ODE

has Lipschitz constant $L = 0$. Note that an approximation technique of higher consistency order, e.g. Runge-Kutta techniques would significantly improve this $\varepsilon(\#\mathcal{I})$.

In Table 8.1, the runtimes for the preparation of the graph including flattening and the calculation of the data flow models in milliseconds are given. In Table 8.2, the runtimes for partitioning and the runtimes of the Verification Engine are provided. We have performed the experiments on a MacBook Pro with a 2.7 GHz Intel Core i7 and 16 GB of RAM. The runtimes are averaged over 10 runs. The long runtimes in the partitioning of the F14 model can be explained by the protoype implementation and a large feedback loop occurring in the model. A more efficient implementation utilising graph packages would be expected to reduce the runtimes there.

Overall, the verification has been successfully performed in less than 6 minutes. The overall complexity of our automation is exponential. This is due to our calculation of the data flow models, which yields $2^{\#C}$ models to check for equivalence, where $\#C$ is the amount of control flow elements. However, apart from this, the equivalence check is expected to scale well for larger, industrially-sized models. We leave the setup of an industrially-sized case study to future work.

8.3 Discussion

In this section, we go through various items to discuss some limitations and strengths of our approach in general and our automation and implementation in particular. Moreover, we assess the computational complexity of our approach.

8.3.1 Limitations and Opportunities

We start with a discussion of the strengths and weaknesses of our automation approach.

Semantic Gap to Simulink

Our formal foundation is the operational semantics [BC12]. We have based our approach on this operational semantics to obtain a sound formal foundation. The semantics of Simulink is not formally defined. In [BC12] as well as in our argumentation, the soundness with Simulink is made plausible via informal argumentation or by providing supplemental arguments that this is consistent with standard mathematical techniques. However, there still exists a semantical gap due to the following reasons.

1. The operational semantics [BC12] is a synchronous semantics, i.e., basically all blocks are evaluated at the same time, i.e., in parallel. As discussed in [RG14], this is not conform to the simulation semantics of Simulink. The simulation semantics in Simulink is a serial one.

2. The operational semantics does not consider numerical errors that occur in reality because it is executed on a machine with a certain precision. We therefore also exlude these numerical errors in our thesis. We focus on errors caused by the approximation procedure in continuous models, which are of greater magnitude and therefore of greater importance.

3. The operational semantics does not consider all optimisation techniques Simulink applies. It is also not possible to extract details from the informal descriptions from [TM17b]. However, it is likely also from the informal descriptions that there are several additional techniques that increase the precision.

To make this semantical gap transparent, we have evaluated the semantical gap to Simulink by implementing the operational semantics and comparing the observations of runs with this implementation and Simulink in the context of a Bachelor thesis [Wor16]. All experimental results revealed negligible deviations in the magnitude of approximately 10^{-10}. As a result, the residual semantic gap can be considered as insignificant.

Remarks on Completeness of the Automation Approach

Besides the limitations presented in Section 4.1, we want to discuss some additional issues related with our automation approach. The limitations and assumptions from the mentioned sections ensure that our algorithm is sound. However, there has been no comprehensive discussion of completeness yet. Completeness in general is impossible due to Rice's theorem [HMU01]. This theorem states that it is impossible to decide if a Turing machine calculates a given function. This implies that it is not possible to decide if two arbitrary machines behave equivalently. Hence, every approach aiming for behavioural equivalence of models is only able to cover a subset of all possible models.

Our automation approach and our implementation are able to verify behavioural equivalence for various relevant models as we have shown in our experimental results. However, we would like to emphasise that we have imposed some restrictions that on the one hand limit the set of correct refactorings that can be detected properly, but on the other hand allow for large automation. Our approach is sound.

Hence, our approach never returns *correct refactoring* in cases where a refactoring is incorrect, i.e., false positives are not possible. However, we may obtain false negatives, i.e., correct refactorings are labelled as being incorrect. This may happen if the partitioning does not assemble equivalent parts in source and target model together. We have provided a fully automated partitioning. However, in some cases, this may yield partitionings whose symbolic equivalence cannot be verified. Such a result is therefore always a motive for further investigation to the user: Either the refactoring is actually incorrect or potentially it is correct, but our approach was not able to detect it.

A particular limitation is caused by the way we have defined our partitioning. In short, we assemble cyclic parts and DAG (directed acyclic graph) parts together. We do this for the following reasons:

1. As we have seen in Section 5.4, (Discrete) Transfer Functions can equivalently be understood as ODEs. In most cases, these ODEs are self-referring, i.e., of the form $\frac{d}{dt}y = f(t, y)$ and therefore associated with a cyclic submodel. A major source for refactorings is an alternative expression of an ODE as Transfer Function (to reduce structural complexity) or vice versa (to reduce functional coomplexity).

2. With our partitioning, we support the occurrence of both time-discrete and time-continuous submodels in a model, i.e., we support a subset of time-hybrid Simulink models. Our approach assumes that within a partition, the submodel is either purely time-discrete or purely time-continuous. However, it is possible that the paradigms change in different partitions.

3. Our partitioning allows to split the model into components that can be verified separately. The AR expressions therefore do not get too large. Our approach therefore improves scalability of the verification.

The first item is an assumption on the model structure. We have observed that it is often fulfilled in models we obtained from our industrial partners. Feedback loops occur quite often in industrial models. They are either modelled as cyclic submodels or Transfer Functions. The output is often extracted from a single point of the cyclic submodel. Counter examples of this observation are cases like the one depicted in Figure 8.5. There, the output is taken from two points from the model: One exit point is located directly after the Add block, one after the Integrator in the source model. This yields in a refactored target model with two partitions: One partition containing the Transfer Function, while the other partition contains the Gain block and the Add block. However, the source model contains only the feedback loop in one partition.

Then, our approach would return *false* because it cannot verify equivalence for all partitions in the target model. This is a counterexample for completeness. In our case study this issue for the example 8.5 did not come up because both submodels were part of a larger feedback loop. However, consider the models were exactly as depicted in the figure.

An expert would be able to perform the partitioning based on his experience, and therefore would be able to avoid such false negatives. However, we have aimed for a largely automated solution, and our approach is able to demonstrate that this is conceptually possible. We leave improvements for the partitioning for future work.

8.3.2 Complexity Assessment

In this section, we assess the complexity of our approach. We do not guarantee that our assessment yields the best complexity assessment possible. This means, parts of our automation potentially could be improved. Since our approach is largely modular, we explicitly support such an approach. Our performance assessment is based on our current implementation. We go through each step, evaluate the complexity, and reason on its correctness.

Graph Representation Calculation

In the first step, we 1) calculate our graph representation from the MeMo Simulink IR in Java, and 2) flatten the submodel hierarchy. For the first part, the graph representation calculation, we visit each block and each signal line. The resulting graph is $G = (B, E, I, O, S, C, port, fun, p)$. The computational effort is $\mathcal{O}(\#E)$ as we traverse the model only once. It suffices to consider $\#E$ because $\#E > \#B$ since we do not have isolated blocks, i.e., each block has at least one ingoing or outgoing edge.

The flattening erases subsystems, and inports and outports that enter or leave subsystems. Moreover, we re-wire the model properly. Again, we only traverse the model once. We therefore obtain the overall complexity

$$\mathcal{O}(\#E).$$

Data Flow Model Calculation

In the Data Flow Model Calculation step, we remove control flow elements, and for each control flow element, we split the given model into two data flow models. Moreover, we associate each data flow model with the configuration. Hence, we

obtain overall

$$\mathcal{O}(2^{\#C}).$$

Partitioning of a Data Flow Model

The partitioning has three steps. Firstly, the user defined partitioning. This yields a complexity of $\mathcal{O}(\#B)$ because the user may at most define $\#B$ partitions. The second part is the partitioning into cyclic and non-cyclic parts yielding a DAG graph. The cycle detection is of complexity $\mathcal{O}(\#B^2)$ because in the cycle detection, we take each pair of vertices (v_1, v_2) and determine is there a path from v_1 to v_2 and vice versa. After the cycle detection, in the third step of the partitioning, the assembly of singleton partitions to DAG parts takes $\mathcal{O}(\#B)$. The final enrichment of the partitioning just adds interface vertices to partitions, which yields $\mathcal{O}(\#B)$. This yields a total complexity of

$$\mathcal{O}(\#B^2).$$

AR Extraction

AR extraction is performed by visiting each vertex once for each equation type (output and state equation). This yields a complexity of

$$\mathcal{O}(\#B).$$

Symbolic Equivalence Checking

Symbolic equivalence checking checks for each data flow model with the same configuration in source and target model for symbolic unifiability. Within data flow models, the equivalence check is performed partition-wise. Let us denote the complexity of the symbolic unification as $\mathcal{O}(\mu_{sv})$ (for Mathematica, symbolic verification). Unfortunately, there is no public information available that gives us a hint about the complexity of the algorithms in Mathematica performing algebraic transformations and unification. As discussed in Section 7.3, we verify

$$f_1(\boldsymbol{y}, \boldsymbol{i}) = f_2(\boldsymbol{y}, \boldsymbol{i})$$

where the $\boldsymbol{y}$ are replaced by the respective Laplace transformed or z-transformed expressions for the state variables. The bold variables are vectors. If we have multiple input variables, i.e., $\boldsymbol{i} = (i_1, ..., i_n)^T$, then we do not know how they match between source and target model. Note that the order of input variables in partitions does not neecessarily to be the same in source and target model. To obtain

an automated solution, we have overcome this problem by permuting all input variables until a suitable unification has been found. If no unification can be found for all permutations, the check fails. For the symbolic verification of a partition p, we obtain $\mathcal{O}(\#I_p! \cdot \mu_{sv})$, where $\#I_p$ is the amount of input interface variables of the partition. The complexity $\mathcal{O}(\mu_{sv})$ is multiplied with $\mathcal{O}(\#I_p!)$ because the equivalence check is performed for all permutations. This in turn can be estimated to $\mathcal{O}(\#B! \cdot \mu_{sv})$, which is a rough estimate because the amount of interface input variables is always below the amount of all blocks.

We perform symbolic equivalence checking for all possible pairs of partitions. The user is asked to confirm that the pairs with verification result *true* are the correct matches in the structure of the partitioning. The amount of partitions can be estimated by the amount of blocks. Hence, we obtain for the symbolic equivalence check of one data flow model the complexity of $\mathcal{O}(\#B! \cdot \#B^2 \cdot \mu_{sv})$. If the user matches the input signals per partition manually, the complexity reduces to $\mathcal{O}(\#B \cdot \mu_{sv})$.

For the symbolic equivalence check for all data flow models, we compare data flow models with the same configuration. We identify the configurations in source and target model via the conditions at the Switch blocks. We have to check all $2^{\#C}$ data flow models. This yields an overall complexity of

$$\mathcal{O}(2^{\#C} \cdot \#B! \cdot \#B^2 \cdot \mu_{sv})$$

Epsilon Calculation and Propagation

The epsilon calculation is an evaluation of a formula in the CAS. This in turn depends on the internal algorithms Mathematica uses for calculation, which is unknown. Let us denote it with $\mathcal{O}(\mu_{cal})$. The epsilon propagation solves ODEs and difference equations for subsequent partitions, and performs evaluations of formulas, which we also estimate by $\mathcal{O}(\mu_{cal})$. This is performed for all partitions and all data flow models. Overall, we obtain

$$\mathcal{O}(2^{\#C} \cdot \#B \cdot \mu_{cal}).$$

Overall Complexity

The steps are performed sequentially. The overall complexity is therefore the sum of all partial complexities, which can further be estimated to the maximum of the partial complexities. The overall complexity is therefore

$$\mathcal{O}(\#E + 2^{\#C} \cdot \#B! \cdot \#B^2 \cdot \mu)$$

Here, we have taken μ as the maximum of both μ_{sv} and μ_{cal}.

Note that for models without control flow, the complexity substantially reduces because we only need to consider two models (source and target model) rather than $2^{\#C}$ data flow models.

8.4 Summary

In this chapter, we have described our implementation, experimental results, and complexity of our automation approach. Our implementation covers the most important aspects of our automation approach as described in Chapter 7. We have described our case studies: a time-continuous F14 model, a hybrid water tank model, and a hybrid cruise control model. In these models, we have performed various refactorings we support with our methodology. We have exchanged submodels representing ODEs by analytical solutions, ODEs by Transfer Function blocks, and performed arithmetic changes. With our approach, we were able to successfully verify equivalence in under 6 minutes in each case.

Our automation approach is of exponential complexity. The exponentiality is due to our calculation of data flow models, which generates two models to be checked for equivalence per control flow element. However, the equivalence check and the epsilon calculation is expected to be well scalable for industrially sized models. This holds particularly for models with few control flow elements, which also occur in the industries.

Our approach is not complete. We may obtain false negatives. This is due to how we defined our partitioning, which may yield different amounts of partitions. In such cases, not for all partitions can be found a suitable equivalent partner. This is the price for a fully automated approach. User guidance during partitioning would improve this.

Chapter 9

Conclusion and Future Work

9.1 Conclusion

In this thesis, we have presented an approach to verify the correctness of refactorings on hybrid data flow-oriented or signal-oriented models. By correctness, we understand behavioural equivalence of source and refactored target model. While there are various approaches tackling behavioural equivalence of hybrid control systems and modelling languages, there is none that considers the approximate nature of such models that is present for executions of hybrid models on machines with finite execution time and storage space. We have closed this gap in this thesis and have provided an approach capable of showing behavioural conformance on hybrid signal-oriented models being modelled and executed on industrially relevant languages. We have chosen Simulink as major representative of signal flow-oriented hybrid control modelling languages to demonstrate our approach.

With the operational semantics for Simulink [BC12], we have based our approach on a formal foundation. With our Abstract Representation, we have defined a denotational semantics for Simulink that is sound with the operational semantics. Both semantics consider the approximate nature of data flow-oriented industrially relevant languages. With our adaptation of the approximate bisimulation to Simulink, we have defined an equivalence notion that is able to cope with the approximate nature of Simulink's semantics. It is based on a well-founded, formal equivalence notion. We have provided sufficient conditions that allow formal verification of behavioural equivalence for a large variety of Simulink models. Moreover, we have provided means to calculate the precision ε, which is the maximal distance for values of source and refactoreed target model that are evaluated by their semantics.

In addition to our approach for formal verification of behavioural equivalence, we have provided an automation approach for a subset of Simulink models. Our automation approach is largely automated. We have implemented our approach. Its complexity is in general exponential. However, exponential runtime is mainly

imposed for models with many control flow elements. The verification and epsilon calculation is well scalable for many indutrially-sized models. Our approach is therefore very well applicable for industrially-sized models with few control flow elements. We have successfully applied our automation approach to three case studies: a model for an F-14 aircraft, a water tank model, and a cruise control model. Verification has been successfully performed in under 6 minutes.

In essence, we have successfully met our criteria as introduced in Chapter 1.

9.2 Future Work

Our methodology for behavioural equivalence of signal flow-oriented modelling languages opens up many interesting future research directions.

Our approach could be improved. We briefly describe three ideas for further improvement.

Firstly, our approach could be extended to cope with refactorings altering the control flow. Examples for refactorings changing control flow could be elimination of data flow paths because they are not reachable or changes of conditions at control flow elements, combined with changing data flow and control flow signals.

Secondly, our calculation of the precision ε of the approximate bisimulation could be improved. There are approaches that provide better estimations of the error during approximation, e.g. [NH04]. The idea there is that local discretisation errors under specific circumstances can eliminate each other and therefore the global discretisation error, which we calculate to obtain our precision ε, does not evolve as strong as we predict. Another interesting aspect is that our ε currently increases with simulation time, i.e., the length of the interval in which the evaluation of the semantics is performed. It would be interesting to examine conditions under which ε can be bounded globally, i.e., for simulation time going to infinity. Theories from both stability of ODEs and numerical stability apply for a consideration of this topic. In this direction also aims a consideration of numerical errors due to floating point arithmetic, which we have excluded in this thesis. As we have pointed out in this thesis, a decreasing sample step size reduces the error ε. However, according to [SWP12], this only applies to a certain minimal value, which depends on the chosen approximation technique. If sample the step size is further decreased, the numerical floating point errors exceed the errors of the approximation to calculate the solutions of the ODEs. These numerical dependencies should be further investigated to improve the ε estimate.

Thirdly, our automation approach could be extended to cope with arbitrary types of ODEs and difference equations. The challenge for non-linear ODEs or difference

equations is that an automated approach as ours that exploits laws applicable to Laplace transform or z-transform, is not attainable. Equivalence proofs for non-linear equations often require manual, complex investigation and proofs. However, our automation approach could be formalised in a theorem prover, and at least provide mechanised proof capability for such cases.

Apart from improvements to our approach, many interesting future directions are conceivable. Our approach compares two models and verifies their behavioural equivalence. Hence, our methodology is applied *after* a refactoring has been performed. It would however be beneficial for a developer to obtain suggestions for potential refactorings for a given model. Based on our methodology, an approach could be developed that provides suggestions for refactorings together with a certificate that guarantees approximate bisimulation up to a user-defined precision ε if the refactoring is applied.

Moreover, an approach that significantly widens the notion of refactoring to model transformations that are only approximately conform even at the idealised mathematical level of our denotational AR would be an interesting research direction. Examples for refactorings in this regard would be the replacement of a model representing a complex solution by a polynomial solution. Mathematically, this is possible due to the Weierstraß theorem, which guarantees that every continuous function can be approximated by a polynomial that varies around the function by at most a given threshold. This would introduce an additional disturbance to our ε. Another interesting suggestion is to aim in the direction of security. State-of-the-art security approaches are mainly considering low-level implementations. However, considering security aspects in early modelling phases would be beneficial. To this end, an approach with the following elements would be interesting to pursue:

Firstly, a formal specification language for security on model level. Secondly, an approach for verification of preservation of such security properties under model transformation, e.g. under refactorings or other, more general model transformations. Thirdly, an approach to verify preservation of security properties under model-to-code transformations. This last part could then potentially close the gap to existing low level security approaches.

Appendix A

Proofs

Proofs from Chapter 5

Lemma 1 (Soundness of AR for Time-discrete Blocks)**.** *Let* $M = (B, E, I, O, S, \emptyset, port, fun, \emptyset)$ *be a time-discrete Simulink model that is being evaluated in the interval* $\mathcal{I}$ *following the operational semantics [BC12], time-discrete blocks with sample step size* h*, simulation start at* t_0*. Let furthermore be* $b \in B$*,* $[\![M]\!]^{\varphi}$ *the AR under an assignment* φ*, and* $out(t) = f(\boldsymbol{i}(t))$ *be the equation associated with block* b*. Then the following holds.*

$$\forall t \in \mathcal{I} : [\![M]\!]^{\varphi}_{OS}(b)(t) = f(\varphi(\boldsymbol{i})(t))$$

Proof. For this proof, we recall the global simulation rule. The global simulation rule is

$$\frac{\sigma(t) \leq \pi(t_{end}) \qquad Eq, \pi \vdash \sigma \xrightarrow{\mathrm{M}} \sigma_1 \qquad Eq, \pi \vdash \sigma_1 \xrightarrow{\mathrm{u}} \sigma_2 \qquad Eq, \pi \vdash \sigma_2 \xrightarrow{\mathrm{s}} \sigma'}{Eq, \pi \vdash \sigma \rightarrow \sigma'}$$

Since in this lemma, we only have time-discrete, arithmetic, input, and output blocks, the solver transition does not apply. We only need to consider major step transitions and update transitions. We distinguish by block types.

First case: if $b \in I$: Then, we have one equation $l_{out_n} = l_{in_n}$. From Definition 41, we know that input variables have been replaced by the sequence determined by the incoming signal from the assignment φ, i.e., we have $l_{out_n} = \varphi(in)_n$. According to [BC12], the input block is associated with a syntactical equation $l_{out} :=_S l_{in}$, where the set S holds the sample steps, i.e., $S = \{t_0 + nh | 0 \leq n \leq N-1\}$, where $t_0 + h(N-1)$ is the end of the simulation interval.

For this formula from the operational semantics $l_{out} :=_S l_{in}$, only the major step transition ($M-$) applies. This says

$$\frac{\langle Eq(M), \sigma\rangle \xrightarrow{o} \sigma'}{Eq, \pi \vdash \sigma \xrightarrow{M} \sigma'}$$

Hence, we need to check the output (o-) transitions that calculate the outputs of blocks. They say

$$\frac{\sigma(t) \notin S}{\langle l_{out} :=_S l_{in}, \sigma\rangle \xrightarrow{o} \sigma} \quad \frac{\sigma(t) \in S \quad \langle l_{in}, \sigma\rangle \xrightarrow{o} r}{\langle l_{out} :=_S l_{in}, \sigma\rangle \xrightarrow{o} \sigma[l \leftarrow r]}$$

The first transition rule says that if $t \in]t_0 + n \cdot h, t_0 + (n+1) \cdot h[$, the value at $[\![M]\!]_{OS}(b) = \sigma(l_{out})$ remains constant. The second transition rule ensures the update at time steps $t_0 + n \cdot h$. The evaluation of $\langle l_{in}, \sigma\rangle \xrightarrow{o} r$ to obtain the update value r is performed via the inference rule

$$\frac{}{\langle l_{in}, \sigma\rangle \xrightarrow{o} \sigma(l_{in})}$$

This means, the value is picked from the assignment, which has just been updated. In summary, this concludes the proof for blocks $b \in I$.

Second case: If $b \in O$: In this case, we have $l_{out_n} = l_{in_n}$. The argumentation is analogously to input blocks. Only major step transitions apply, the output of an output block is an interval-wise constant function that is updated at time steps in S. The input signal, which is defined in the state $\sigma(l_{in})$ of the input signal is fed through.

Third case: If $b \in S$ is a Unit Delay block: In this case, our AR gives us the equation $l_{out_{n+1}} = l_{in_n}$. Regarding the operational semantics, we have a major step transition.

$$\frac{\sigma(t) \notin S}{\langle l_{out} :=_S d, \sigma\rangle \xrightarrow{o} \sigma} \quad \frac{\sigma(t) \in S \quad \langle d, \sigma\rangle \xrightarrow{o} r}{\langle l_{out} :=_S d, \sigma\rangle \xrightarrow{o} \sigma[l \leftarrow r]}$$

On the right hand side of the equations, the internal variable d occurs. The inference rules for the internal variable are the update transitions

$$\frac{d :=_S l_{in} \in Eq \quad \sigma(t) \notin S}{Eq, \pi \vdash \sigma \xrightarrow{u} \sigma} \quad \frac{d :=_S l_{in} \in Eq \quad \sigma(t) \in S}{Eq, \pi \vdash \sigma \xrightarrow{u} \sigma[d \leftarrow \sigma(l_{in})]}$$

The inference rule on the left hand side of the major step transition implies analogously that in the interval $]t_0 + n \cdot h, t_0 + (n+1) \cdot h[$ the output of the block remains constant. This applies to the internal variable d as well. However, as the time steps $t_0 + n \cdot h$ are reached, firstly the value of the internal variable is written to the output, and afterwards the value at the entry of the block is stored in the internal variable. This explains the delay, i.e., the behaviour $l_{out}(t) = l_{in}(t_0 + (n-1) \cdot h)$ for $t \in [t_0 + n \cdot h, t_0 + (n+1) \cdot h$. This is precisely described by the equation for Unit Delay blocks in the AR $l_{out_{n+1}} = l_{in_n}$. Note that we have used the sequences to

abbreviate the interval-wise constant functions. Additionally the second equation $l_{in_0} = init$ ensures the value in the interval $[t_0, t_0 + h[$. Hence, we have shown the soundness for $b \in S$ being a Unit Delay block as well. The proof for delays > 1 works analogously. With the auxiliary variables introduced if delay length is greater than 1, we can reduce this case to the delay $= 1$ case.

Fourth case: b being an arithmetic block: In this case, our AR gives us an equation $l_{out_n} = f(l_{in_{1_n}}, ..., l_{in_{m_n}})$. The operational semantics gives us the major step transition that is applicable.

$$\frac{\sigma(t) \notin S}{\langle l_{out} :=_S f(l_{in_1}, ..., l_{in_m}), \sigma\rangle \xrightarrow{o} \sigma} \quad \frac{\sigma(t) \in S \quad \langle f(l_{in_1}, ..., l_{in_m}), \sigma\rangle \xrightarrow{o} r}{\langle l_{out} :=_S f(l_{in_1}, ..., l_{in_m}), \sigma\rangle \xrightarrow{o} \sigma[l \leftarrow r]}$$

The evaluation in the inference rules of $f(l_{in_1}, ..., l_{in_m})$ to the value r that is then updating the output is ensured by the $o-$ transitions that evaluate expressions.

$$\frac{}{\langle l_{in_i}, \sigma\rangle \xrightarrow{o} \sigma(l_{in_i})} \quad \frac{\langle l_{in_1}, \sigma\rangle \xrightarrow{o} r_1, ... \langle l_{in_m}, \sigma\rangle \xrightarrow{o} r_m}{\langle f(l_{in_1}), ..., l_{in_m}), \sigma\rangle \xrightarrow{o} f(r_1, ..., r_m)}$$

Hence, we have shown the soundness for all block types in time-discrete models. This concludes the proof. □

Lemma 2 (Soundness of AR for Time-continuous Blocks)**.** *Let $M = (B, E, I, O, S, \emptyset, port, fun, \emptyset)$ be a time-continuous Simulink model. that is being executed in the interval $\mathcal{I}$ following the operational semantics [BC12], simluation start at t_0 under the assignment φ. We assume that $Im(\varphi) \subset C(\mathcal{I}, \mathbb{R})$, i.e., the input blocks are instantiated with continuous signals. Moreover, $Im(\varphi)$ are assumed to be piece-wise bounded functions.*

Let furthermore be $b \in S$ an Integrator block, $[\![M]\!]^\varphi$ the AR under an assignment φ, and $\frac{d}{dt}y(t) = in(t)$ the equation of the AR at b.

For a sample step size h, we associate a function $[\![M]\!]^{\varphi,h}_{OS}(b)$ as follows.

$$[\![M]\!]^{\varphi,h}_{OS}(b)(\tau) = \frac{\sigma_{n+1}(v) - \sigma_n(v)}{h}(\tau - t_n) + \sigma_n(v) \text{ for } \tau \in [t_n, t_{n+1}[$$

where $\sigma_n, \sigma_{n+1} \in \Sigma_M$ such that $\sigma_n(t) = t_n$, $\sigma_{n+1}(t) = t_{n+1}$, $t_{n+1} = t_n + h$, and v being the variable associated with block b. This function yields the same values as the execution by the operational semantics $[\![M]\!]_{OS}(b)$ for block b at time steps $t_n = t_0 + n \cdot h$ if the Euler method has been chosen as approximation method by the user in Simulink.

Between time steps. i.e., for $t \in]t_n, t_{n+1}[$, the function follows a linear curve connecting the values at time steps t_n, t_{n+1} rather than being constant as in the trajectory of $[\![M]\!]_{OS}(b)$. We assume that the trajectory of the function $[\![M]\!]^{\varphi,h}_{OS}(b)$ is bounded on $\mathcal{I}$.

Every sequence in the family

$$\mathcal{F} := \left\{ [\![M]\!]^{\varphi,h}_{OS}(b) \middle| h = \frac{\#\mathcal{I}}{n} \text{ for } n \in \mathbb{N}^+ \right\}$$

of evaluations with fixed sample step sizes contains a subsequence that converges uniformly to a function f if $\lim_{n\to\infty} h_n = 0$. For this function the following holds.

$$f \models [\![M]\!]^{\varphi}(b)$$

We also denote this relationship between $[\![M]\!]^{\varphi}_{OS}(b)$ and $[\![M]\!]^{\varphi}(b)$ with

$$[\![M]\!]^{\varphi}_{OS}(b) \models_{h\to 0} [\![M]\!]^{\varphi}(b)$$

Proof. We start with recalling the global simulation rule.

$$\frac{\sigma(t) = \pi(t_{end})}{Eq, \pi \vdash \sigma \to \sigma}$$

$$\frac{\sigma(t) \leq \pi(t_{end}) \qquad Eq, \pi \vdash \sigma \xrightarrow{\text{M}} \sigma_1 \qquad Eq, \pi \vdash \sigma_1 \xrightarrow{\text{u}} \sigma_2 \qquad Eq, \pi \vdash \sigma_2 \xrightarrow{\text{s}} \sigma'}{Eq, \pi \vdash \sigma \to \sigma'}$$

As in Lemma 16, major step transitions ensure that the arithmetic functions are properly evaluated before the solver transition, which is relevant to be considered for time-continuous blocks. Update transitions are not relevant in this context because we do not have time-discrete blocks in time-continuous models.

We recall the solver transition.

$$\frac{Eq, \pi \vdash \sigma \xrightarrow{i} \sigma'}{Eq, \pi \vdash \sigma \xrightarrow{s} \sigma'}$$

The solver transition consists of an integration ($i-$) transition. Note that we have assumed fixed sample step size and therefore, the zero-crossing detection ($zc-$) transition, and the transition to update the sample step size ($h-$ transition) do not apply for our case. The latter two are only applicable for variable step size solvers. The integrator transition applies an approximation method to determine the value for the next sample step.

Hence, the subsequent state at the Integrator stateful block with variable v, i.e., $\sigma_{n+1}(v)$ is calculated as $\sigma_{n+1}(v) = \sigma n(v) + h_n \sigma(v')$ (for the Euler method) where

v' is the variable of the block preceding the stateful block, i.e., the entrance of the stateful block. Since arithmetic functions are evaluated prior the solver transition in the major step transition, we can conclude that at the time steps t_n, we have

$$[\![M]\!]^{\varphi,h}_{OS}(b)(t_{n+1}) = [\![M]\!]^{\varphi,h}_{OS}(b)(t_n) + hf([\![M]\!]^{\varphi,h}_{OS}(\boldsymbol{i})(t_n), [\![M]\!]^{\varphi,h}_{OS}(\boldsymbol{y})(t_n))$$

where the i are the input blocks, the y stateful blocks impacting the Integrator block b.

Between the time steps, i.e., at $]t_n, t_{n+1}[$, we have

$$[\![M]\!]^{\varphi,h}_{OS}(b)(\tau) = \frac{\sigma_{n+1}(v) - \sigma_n(v)}{h}(\tau - t_n) + \sigma_n(v)$$

as defined in this theorem. When we combine this with the result we have just obtained, we can conclude for $\tau \in [t_n, t_{n+1}[$

$$\begin{aligned}[\![M]\!]^{\varphi,h}_{OS}(b)(\tau) &= \frac{[\![M]\!]^{\varphi,h}_{OS}(b)(t_{n+1}) - [\![M]\!]^{\varphi,h}_{OS}(b)(t_n)}{h}(\tau - t_n) + [\![M]\!]^{\varphi,h}_{OS}(b)(t_n) \\ &= f([\![M]\!]^{\varphi,h}_{OS}(\boldsymbol{i})(t_n), [\![M]\!]^{\varphi,h}_{OS}(\boldsymbol{y})(t_n))(\tau - t_n) + [\![M]\!]^{\varphi,h}_{OS}(b)(t_n) \quad \text{(A.1)}\end{aligned}$$

So, we now can focus on showing that for every sequence in the family $\mathfrak{F}$ consisting of functions for which equation A.1 holds and with $\lim_{n\to\infty} h_n = 0$, there exists a uniformly convergent subsequence for which the AR equations hold.

This is achieved analogously to the proof of the theorem of Peano [Aul04]: We apply the theorem of Arzelà-Ascoli. To this end, we need to show 1) that $\mathcal{F}$ is equicontinuous and piece-wise bounded.

First of all, the function $[\![M]\!]^{\varphi,h}_{OS}(b)$ is piece-wise bounded.We have assumed that in the theorem. This assumption simply guarantees that the simulation yields finite values across the simulation interval $\mathcal{I}$, i.e, where the model is being executed.

Secondly, we need to show the equicontinuity of $\mathcal{F}$. Let $\varepsilon > 0$, $t_1, t_2 \in \mathcal{I}$. We distinguish two cases.

1. If t_1, t_2 are in the same interval, i.e., $\exists n : t_1, t_2 \in [t_n, t_{n+1}[$. Then

$$\begin{aligned}|[\![M]\!]^{\varphi,h}_{OS}(b)(t_1) - [\![M]\!]^{\varphi,h}_{OS}(b)(t_2)| = \\ |(t_1 - t_n)f([\![M]\!]^{\varphi,h}_{OS}(\boldsymbol{i})(t_n), [\![M]\!]^{\varphi,h}_{OS}(\boldsymbol{y})(t_n) - \\ (t_2 - t_n)f([\![M]\!]^{\varphi,h}_{OS}(\boldsymbol{i})(t_n), [\![M]\!]^{\varphi,h}_{OS}(\boldsymbol{y})(t_n))| \leq \\ (\max_{t\in\mathcal{I}} [\![M]\!]^{\varphi,h}_{OS})|t_1 - t_2|\end{aligned}$$

Note that $m := \max_{t\in\mathcal{I}} [\![M]\!]^{\varphi,h}_{OS}$ exists per assumption.

2. If t_1, t_2 are in different intervals, i.e., $t_1 \in [t_{n_1}, t_{n_1+1}[, t_2 \in [t_{n_2}, t_{n_2}+1[$ for suitable n_1, n_2 with $n_1 < n_2$. Then

$$\begin{aligned}
|[\![M]\!]_{OS}^{\varphi,h}(b)(t_1) - [\![M]\!]_{OS}^{\varphi,h}(b)(t_2)| \leq \\
|[\![M]\!]_{OS}^{\varphi,h}(b)(t_1) - [\![M]\!]_{OS}^{\varphi,h}(b)(t_{n_1+1})| + \\
\sum_{\nu=n_1+1}^{n_2-1} |[\![M]\!]_{OS}^{\varphi,h}(b)(t_\nu) - [\![M]\!]_{OS}^{\varphi,h}(b)(t_\nu)| + \\
|[\![M]\!]_{OS}^{\varphi,h}(b)(t_{n_2}) - [\![M]\!]_{OS}^{\varphi,h}(b)(t_2)| \leq \\
m(t_{n_1+1} - t_1) + \sum_{\nu=n_1+1}^{n_2-1} (t_{\nu+1} - t_\nu) + (t_2 - t_{n_2})) = \\
m|t_1 - t_2|
\end{aligned}$$

In essence, in both possible cases, we have

$$|[\![M]\!]_{OS}^{\varphi,h}(b)(t_1) - [\![M]\!]_{OS}^{\varphi,h}(b)(t_2)| \leq m|t_1 - t_2|$$

Hence, we can choose $\delta := \frac{\varepsilon}{m}$. Then with $|t_1 - t_2| < \delta$, we obtain

$$|[\![M]\!]_{OS}^{\varphi,h}(b)(t_1) - [\![M]\!]_{OS}^{\varphi,h}(b)(t_2)| \leq m|t_1 - t_2| < m\delta = \varepsilon$$

This concludes the proof of equicontinuity of $\mathcal{F}$. With the theorem of Arzelà-Ascoli, we can conclude that every sequence in $\mathcal{F}$ contains a subsequence that converges uniformly to a function $\psi : \mathcal{I} \to \mathbb{R}$. This means, every sequence of functions $([\![M]\!]_{OS}^{\varphi,h_n}(b))_n$ contains a subsequence $([\![M]\!]_{OS}^{\varphi,h_{n_k}}(b))_k$ such that

$$\lim_{k\to\infty} [\![M]\!]_{OS}^{\varphi,h_{n_k}}(b) = \psi$$

for a function $\psi : \mathcal{I} \to \mathbb{R}$, and the convergence is uniformly.

In the final step, we have to show that if $b \in S$, the function ψ is a solution of the ODE $\frac{d}{dt}y(t) = f(\boldsymbol{i}(t), \boldsymbol{y}(t)), y(t_0) = y_0$, which is the only critical equation in the AR whose validity remains to be shown. To this end, we recall that the ODE can equivalently be formulated as

$$y(t) = y_0 + \int_{t_0}^{t} f(\boldsymbol{i}(\tau), \boldsymbol{y}(\tau))d\tau$$

due to the Fundamental Theorem of Calculus. From our previous argumentation and the *consisteny* of the Euler method (cf. 2.2.5 in the Background chapter), we

obtain immediately

$$\left| \frac{d}{dt} [\![M]\!]_{OS}^{\varphi,h_{n_k}}(b)(t) - f(\boldsymbol{i}(t), \boldsymbol{y}(t)) \right| \xrightarrow{h_{n_k} \to 0} 0$$

Hence, we can conclude that

$$\lim_{h_{n_k} \to 0} \left| y_0 + \int_{t_0}^{t} \frac{d}{dt} [\![M]\!]_{OS}^{\varphi,h_{n_k}}(b)(t)dt - y_0 - \int_{t_0}^{t} f(\boldsymbol{i}(t), \boldsymbol{y}(t))dt \right| =$$
$$\left| \int_{t_0}^{t} \lim_{h_{n_k} \to 0} \frac{d}{dt} [\![M]\!]_{OS}^{\varphi,h_{n_k}} dt - \int_{t_0}^{t} f(\boldsymbol{i}(t), \boldsymbol{y}(t))dt \right| = 0$$

The first equation, which exploits the interchangeability of integral and limit, is possible because of the uniform convergence.

In essence, the limit function of the subsequence in $\mathcal{F}$ being guaranteed by the theorem of Arzelà-Ascoli fulfills the ODE in the AR of our integrator block.

□

Proofs from Chapter 6

Theorem 1 (Behavioural Equivalence for Purely Time-Discrete Models)**.** *Let M_1 and M_2 be purely time-discrete Simulink models that are executed on the interval $\mathcal{I}$ with the same sample step size h, each time-discrete block with the same sample step size h_d. Let furthermore be $(f_n)_n, (g_n)_n$ be sequences for which the following holds for all assignments φ.*

$$(f_n)_n \models [\![M_1]\!]^{\varphi}, (g_n)_n \models [\![M_2]\!]^{\varphi}$$

If

$$\forall 0 \leq n \leq n_{max} : f_n = g_n$$

then

$$M_1 \sim_0 M_2$$

This means, the models are approximately bisimilar with precision 0. In this case, the approximate bisimilarity coincides with traditional bisimulation.

Proof. As we have seen in Chapter 5 in Theorem 16, the AR for time-discrete models is sound with the operational semantics [BC12]. This means, the equations in $[\![M_i]\!]^{\varphi}$ determine the exact behaviour at the output of the models. So, we know that

$$[\![M_i]\!]_{OS}^{\varphi} \models [\![M_i]\!]^{\varphi}$$

We furthermore know that the solutions of $[\![M_i]\!]^\varphi$, say $(\psi_{i_n})_n$, are unique. Note that sequences are always uniquely determined by difference equations, i.e., if a solution exists, it is unique. This is because the difference equation is a recursive definition: Starting from the initial value, the next value in each step can be calculated by applying the arithmetic function on the previous value as defined in the difference equation. This implies $\forall n : \psi_{1_n} = f_n = g_n = \psi_{2_n}$. Moreover, recall that $(\psi_{i_n})_n, (f_n)_n$, and $(g_n)_n$ actually describe interval wise functions, i.e., the functions are constant at intervals $[t_0 + n \cdot h, t_0 + (n+1) \cdot h_d[$. In essence, this implies

$$\|[\![M_1]\!]^\varphi_{OS}(\tau) - [\![M_2]\!]^\varphi_{OS}(\tau)\|_\infty = 0$$

for all $\tau \in \mathcal{I}$ and assignments φ. This concludes the proof.

□

Theorem 2 (Approximate Behavioural Equivalence for Purely Time-Continuous Models)**.** *Let M_1 and M_2 be purely time-continuous Simulink models that are executed on the interval $\mathcal{I}$. Let furthermore be $f, g : \mathcal{I} \to \mathbb{R}$ be functions for which the following holds for all assignments φ with $Im(\varphi) \subset C(\mathcal{I}, \mathbb{R})$.*

$$f \models [\![M_1]\!]^\varphi, g \models [\![M_2]\!]^\varphi$$

If f and g are uniquely determined by the equations of the AR and

$$\forall t \in \mathcal{I} : f(t) = g(t),$$

then there exists $\varepsilon(\#\mathcal{I}) > 0$ such that

$$M_1 \sim_{\varepsilon(\#\mathcal{I})} M_2$$

Proof. We know from Lemma 17 in Chapter 5 that

$$[\![M_i]\!]^\varphi_{OS} \models_{h \to 0} [\![M_i]\!]^\varphi.$$

This means that the sequence of semantical executions by decreasing sample step size h converges uniformly to the solutions of ODEs (if present) that occur in $[\![M_i]\!]^\varphi$, say ψ_i. Since they are unique, we know that $\psi_1(t) = f(t) = g(t) = \psi_2(t)$. So, the solutions solving the ARs yield the same values. However, the operational semantics differs from these exact solutions due to the approximations performed in the solver transitions. We determine the maximal distance between the semantical evaluations of the models, i.e., the maximal distance of the discretisation of these solutions. We

know from the proof of Lemma 17 that at the sample steps t_n the values of the semantical executions at stateful blocks y follow the following rule.

$$[\![M_i]\!]^{\varphi,h}_{OS}(\boldsymbol{y})(t_{n+1}) = [\![M_i]\!]^{\varphi,h}_{OS}(\boldsymbol{y})(t_n) + h\chi_i([\![M]\!]^{\varphi,h}_{OS}(\boldsymbol{i})(t_n), [\![M_i]\!]^{\varphi,h}_{OS}(\boldsymbol{y})(t_n)) \qquad \text{(A.2)}$$

The expressions representing the functions χ_i coincide with the expressions in the ARs $[\![M_i]\!]^{\varphi}(\boldsymbol{y})$ describing the ODEs. Between sample step sizes, i.e., for $t \in [t_n, t_{n+1}[$, the values of $[\![M_i]\!]^{\varphi,h}_{OS}(\boldsymbol{y})$ are constant. The formula A.2 assumes that the chosen approximation technique is Euler. In general, we have

$$[\![M_i]\!]^{\varphi,h}_{OS}(\boldsymbol{y})(t_{n+1}) = [\![M_i]\!]^{\varphi,h}_{OS}(\boldsymbol{y})(t_n) + h\Phi(h, \chi_i)(t_n, [\![M_i]\!]^{\varphi,h}_{OS}(\boldsymbol{i}(t_n), [\![M_i]\!]^{\varphi,h}_{OS}(\boldsymbol{y}(t_n))))$$

with a processing function Φ. In any case, Lemmas 14 and 15 from the Background Chapter 2 provide a global error $\varepsilon_i(\#\mathcal{I})$ for the distance

$$\|[\![M_i]\!]^{\varphi,h}_{OS}(\boldsymbol{y}) - \psi_i\|_\infty \leq \varepsilon_i(\#\mathcal{I})$$

This also applies to the actual semantical evaluation

$$\|[\![M_i]\!]^{\varphi}_{OS}(\boldsymbol{y}) - \psi_i\| \leq \varepsilon_i(\#\mathcal{I})$$

that keeps the values of the variables constant between time steps. Hence, the maximal distance between the semantical evaluations is

$$\|[\![M_1]\!]^{\varphi}_{OS}(\boldsymbol{y}) - [\![M_2]\!]^{\varphi}_{OS}(\boldsymbol{y})\| = \|[\![M_1]\!]^{\varphi}_{OS}(\boldsymbol{y}) + \psi_1 - \psi_2 - [\![M_2]\!]^{\varphi}_{OS}(\boldsymbol{y})\| \leq$$

$$\|[\![M_1]\!]^{\varphi}_{OS}(\boldsymbol{y}) - \psi_1\| + \|[\![M_2]\!]^{\varphi}_{OS}(\boldsymbol{y}) - \psi_2\| \leq \varepsilon_1(\#\mathcal{I}) + \varepsilon_2(\#\mathcal{I})$$

with triangular equation.

Hence, we have obtained

$$M_1(\boldsymbol{y}) \sim_{\varepsilon_1(\#\mathcal{I})+\varepsilon_2(\#\mathcal{I})} M_2(\boldsymbol{y})$$

Thus, the output at stateful blocks in both models deviates or is disturbed by the maximal magnitude of $\varepsilon(\#\mathcal{I}) := \varepsilon_1(\#\mathcal{I}) + \varepsilon_2(\#\mathcal{I})$.

To obtain an estimate for the overall model, we need to transfer this result to the output of the models. The semantical evaluation of the output is expressed by an equation

$$[\![M_i]\!]^{\varphi}_{OS}(t) = r([\![M_i]\!]^{\varphi}_{OS}(\boldsymbol{i})(t), [\![M_i]\!]^{\varphi}_{OS}(\boldsymbol{y})(t))$$

for a suitable function r (the output equation). This r modifies the disturbance $\varepsilon(\#\mathcal{I})$

that is located at the variables $\boldsymbol{y}$ according to the occurring arithmetic functions. For instance if $r(y_1, y_2) = y_1 + y_2$, where both stateful variables y_i are disturbed by $\varepsilon(\#\mathcal{I})$ as just discussed, then the resulting deviation is $\varepsilon(\#\mathcal{I})+\varepsilon(\#\mathcal{I}) = 2\varepsilon(\#\mathcal{I})$. For other arithmetic functions, the resulting deviation can be calculated analogously. In essence, we obtain a maximal deviation of the semantical evaluations of the models, which concludes the proof.

□

Theorem 3 (Approximate Bisimulation for Hybrid Models)**.** *Let M_1, M_2 be two hybrid Simulink models with the same set of configurations X, evaluated in simulation interval $\mathcal{I}$ with the same sample step size. Moreover, for all assignments φ, let*

$$\forall \chi \in X : f_{1_\chi} = f_{2_\chi} \text{ for } f_{i_\chi} \models [\![M_{i_\chi}]\!]^\varphi.$$

This means, the solutions of our AR yield all the same values for all output signals including the control signals. Let all state equations in the AR be uniquely and globally solvable (which is for difference equations always the case, for differential equations fulfilled if the right hand sides of the equations are globally Lipschitz continuous). Let all data flow models be purely time-discrete or purely time-continuous, i.e., no data flow model time-hybrid. Moreover, let

$$\forall \chi \in X : cond_1(\chi) = cond_2(\chi).$$

This means, all conditions associated with the configurations are the same. Let $\varepsilon_\chi(\#\mathcal{I})$ be the numbers such that

$$M_{1_\chi} \sim_{\varepsilon_\chi(\#\mathcal{I})} M_{2_\chi},$$

i.e., $\varepsilon_\chi(\#\mathcal{I})$ are the precisions of the approximate bisimulation for the data flow models with the same configurations. Then

$$M_1 \sim_{\varepsilon(\#\mathcal{I})} M_2$$

with

$$\varepsilon(\#\mathcal{I}) := \max_{\chi \in X} \varepsilon_\chi(\#\mathcal{I})$$

if no time-delayed switching as discussed in the previous section occurs. Otherwise, $\varepsilon(\#\mathcal{I}) = \max_{\chi \in X} \varepsilon_\chi(\#\mathcal{I}) + \max_{t \in \mathcal{I}, \chi \in X} \|f_{1_\chi} - f_{2_\chi}\|_\infty$.

Proof. The conditions for Theorems 4 and 2 are fulfilled because data flow models are models without control flow elements, and the trajectories have the same values (uniqueness of solutions of the AR is assumed in the theorem). We therefore obtain ε_χ for each configuration χ. According to the interpretation of hybrid models 42, the

time steps at which the respective configuration is fed through, is determined by the condition, and the trajectory of the control signals, i.e., configuration χ applies if

$$\models cond(\chi)[b_{control_j} \leftarrow f_{control_{j_i}}] \text{ for } f_{control_{j_i}} \models [\![M_\chi]\!]^{\varphi;t_{s_i},f_{y_i}(f_{i-1}(t_{s_i}),\varphi(i)(t_{s_i}))}(b_{control_j}).$$

In the formula, the index i was the number of the partial interval $\mathcal{I}_i$ of $\mathcal{I}$. Since the conditions in both models are the same, and the trajectories of the control signals as well (we have defined the blocks at the control ports $b_{control_j}$ as output blocks and therefore, the trajectories yield the same values for them as well per assumption), the time steps at which switches between configurations occur must be the same in both models. The only divergence may occur due to time-delayed behaviour, for which we have provided additional conditions. Also, the initial values of $f_i : \mathcal{I}_i \to \mathbb{R}$ with

$$f_i \models [\![M_\chi]\!]^{\varphi;t_{s_i},f_{y_i}(f_{i-1}(t_{s_i}),\varphi(i)(t_{s_i}))}, f_0 \models [\![M]\!]_\chi^{\varphi,t_0,y_0}$$

are the same at the beginning of the partial intervals because the time step of the switch determines their value, which is the same per assumption. Note that it is sufficient to check equality of solutions, i.e., $f_{1_\chi} = f_{2_\chi}$ for solutions with initial value t_0, y_0, which are given as parameters by the user. This suffices because the state equations are fix and determined by the data flow models: Each data flow model contains exactly one possible state equation the semantical evaluation pursues. The time steps where the switches between data flow models occur are unknown, but equal per assumption. Since we have assumed global uniqueness for the solutions, the solutions after a switch are in turn equal. It remains to be shown that $\varepsilon = \max_{\chi \in X} \varepsilon_\chi$ is properly calculated. Each data flow model that is possible is represented. Hence, all possible disturbances ε_χ have been considered. Let t_s be a switch from configuration χ_1 to χ_2. Then, the initial value in the next interval of semantical evaluation can be disturbed by ε_{χ_1}. However, $\varepsilon_{\chi_1} \leq \varepsilon$ and $\varepsilon_{\chi_2} \leq \varepsilon$, and

$$\varepsilon(\#\mathcal{I}) = \max_{\chi \in X} \varepsilon_\chi = f_{err}\left(h, \#\mathcal{I}, \max_{\chi \in X} L_\chi, \max_{\chi \in X, t \in \mathcal{I}} \left\| \frac{d}{dt} f_\chi(t) \right\|_\infty\right).$$

From our numerical preliminaries in Section 2.2.5 of the Background chapter, we know that the formula for f_{err} considers numerical stability, i.e., errors from sample steps prior t_s have been considered in the formula. Hence, ε already includes the change of the initial value at time step t_s because it exceeds both ε_{χ_1} and ε_{χ_2}. Note that we have excluded time-hybrid data flow models, i.e., there are no switches between time-discrete to time-continuous data flow models possible that share common state variables. This justifies the choice of $\varepsilon(\#\mathcal{I})$.

If time-delays may occur, then approximate bisimulation holds outside of these intervals with the precision ε as well. However, within the intervals of time-delayed behaviour, we need to add the maximal distance of the tranjectories because this may be the maximal distance between the values of the semantical evaluation. □

Bibliography

[ABDM00] Eugene Asarin, Olivier Bournez, Thao Dang, and Oded Maler. Approximate reachability analysis of piecewise-linear dynamical systems. In *International Workshop on Hybrid Systems: Computation and Control*, pages 20–31. Springer, 2000.

[ABSH11] Bakr Al-Batran, Bernhard Schätz, and Benjamin Hummel. Semantic clone detection for model-based development of embedded systems. In *Model Driven Engineering Languages and Systems*, pages 258–272. Springer, 2011.

[ACHH93] Rajeev Alur, Costas Courcoubetis, Thomas A Henzinger, and Pei-Hsin Ho. Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems. In *Hybrid systems*, pages 209–229. Springer, 1993.

[AG08] Mohammed A Al-Gwaiz. *Sturm-Liouville theory and its applications*. Springer, 2008.

[AHH96] Rajeev Alur, Thomas A. Henzinger, and Pei-Hsin Ho. Automatic symbolic verification of embedded systems. *IEEE Transactions on Software Engineering*, 22(3):181–201, 1996.

[AHLP00] Rajeev Alur, Thomas A. Henzinger, Gerardo Lafferriere, and George J. Pappas. Discrete abstractions of hybrid systems. *Proceedings of the IEEE*, 88(7):971–984, 2000.

[AKRS08] Rajeev Alur, Aditya Kanade, S Ramesh, and KC Shashidhar. Symbolic analysis for improving simulation coverage of simulink/stateflow models. In *Proceedings of the 8th ACM international conference on Embedded software*, pages 89–98. ACM, 2008.

[AMF14] Houssam Abbas, Hans Mittelmann, and Georgios Fainekos. Formal property verification in a conformance testing framework. In *Proceedings of the 12th ACM-IEEE International Conference on Formal Methods and Models for System Design (MEMOCODE'14)*, pages 155–164. IEEE, 2014.

[Arn84] Herbert Arndt. Numerical solution of retarded initial value problems: local and global error and stepsize control. *Numerische Mathematik*, 43(3):343–360, 1984.

[ASK04] Aditya Agrawal, Gyula Simon, and Gabor Karsai. Semantic translation of simulink/stateflow models to hybrid automata using graph transformations. *Electronic Notes in Theoretical Computer Science*, 109:43–56, 2004.

[Ass17] Modelica Association. *Modelica® - A Unified Object-Oriented Language for Systems Modeling, Language Specification Version 3.4*. Modelica Association, 2017.

[Aul04] Bernd Aulbach. *Gewöhnliche Differenzialgleichungen: Hier auch später erschienene, unveränderte Nachdrucke*. Elsevier Spektrum Akademischer Verlag, München, 2004.

[Bas16] Johannes Basler. Implementierung eines Checkers zum Nachweis der Korrektheit von Refactorings auf diskreten und kontinuierlichen Simulink Modellen. Bachelor thesis, Technische Universitaet Berlin, 2016.

[BBH+14] Peter Braun, Manfred Broy, Frank Houdek, Matthias Kirchmayr, Mark Müller, Birgit Penzenstadler, Klaus Pohl, and Thorsten Weyer. Guiding requirements engineering for software-intensive embedded systems in the automotive industry. *Computer Science-Research and Development*, 29(1):21–43, 2014.

[BC12] Olivier Bouissou and Alexandre Chapoutot. An operational semantics for simulink's simulation engine. *ACM SIGPLAN Notices*, 47(5):129–138, 2012.

[BCC+17] Timothy Bourke, Francois Carcenac, Jean-Louis Colaço, Bruno Pagano, Cédric Pasteur, and Marc Pouzet. A synchronous look at the simulink standard library. *ACM Transactions on Embedded Computing Systems (TECS)*, 16(5s):176, 2017.

[Bos11] Pontus Boström. Contract-based verification of simulink models. In *Formal Methods and Software Engineering (ICFEM 2011)*, pages 291–306. Springer, 2011.

[Bra98] Michael S Branicky. Multiple lyapunov functions and other analysis tools for switched and hybrid systems. *IEEE Transactions on automatic control*, 43(4):475–482, 1998.

[But05] John Charles Butcher. *Numerical methods for ordinary differential equations*. Wiley, Chichester, 2005.

[But16] John Charles Butcher. *Numerical methods for ordinary differential equations*. John Wiley & Sons, 2016.

[CK99] Alongkrit Chutinan and Bruce H Krogh. Verification of polyhedral-invariant hybrid automata using polygonal flow pipe approximations. In *International workshop on hybrid systems: computation and control*, pages 76–90. Springer, 1999.

[CK03] Alongkrit Chutinan and Bruce H. Krogh. Computational techniques for hybrid system verification. *IEEE Transactions on Automatic Control*, 48(1):64–75, 2003.

[CLRS01] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. Introduction to algorithms second edition, 2001.

[CMKSP10] T Erkkinen Conrad, T Maier-Komor, G Sandmann, and M Pomeroy. Code generation verification–assessing numerical equivalence between simulink models and generated code. In *4th Conference Simulation and Testing in Algorithm and Software Development for Automobile Electronics.*, 2010.

[Con10] John B. Conway. *A course in functional analysis*, volume 96 of *Graduate texts in mathematics*. Springer, New York, 2010.

[CRH93] Zhou Chaochen, Anders P Ravn, and Michael R Hansen. An extended duration calculus for hybrid real-time systems. In *Hybrid Systems*, pages 36–59. Springer, 1993.

[Dan16] Armin Danziger. Implementierung einer grafischen Lösung zur Darstellung von MATLAB / Simulink Modellen. Bachelor thesis, Technische Universitaet Berlin, 2016.

[DHJ+10] Florian Deissenboeck, Benjamin Hummel, Elmar Juergens, Michael Pfaehler, and Bernhard Schaetz. Model clone detection in practice. In *Proceedings of the 4th International Workshop on Software Clones*, pages 57–64, New York, 2010. ACM.

[DM07] Alexandre Donzé and Oded Maler. Systematic simulation using sensitivity analysis. *Hybrid Systems: Computation and Control*, pages 174–189, 2007.

[EA06] Refaat El Attar. *Lecture notes on Z-Transform*, volume 1. Lulu. com, 2006.

[EP06] Abbas Edalat and Dirk Pattinson. Denotational semantics of hybrid automata. In *Foundations of Software Science and Computation Structures*, volume 3921 of *LNCS*, pages 231–245, 2006.

[EVZH07] Chance Elliott, Vipin Vijayakumar, Wesley Zink, and Richard Hansen. National instruments labview: a programming environment for laboratory automation and measurement. *JALA: Journal of the Association for Laboratory Automation*, 12(1):17–24, 2007.

[FE98] Peter Fritzson and Vadim Engelson. Modelica—a unified object-oriented language for system modeling and simulation. In *European Conference on Object-Oriented Programming*, pages 67–90. Springer, 1998.

[FLGD+11] Goran Frehse, Colas Le Guernic, Alexandre Donzé, Scott Cotton, Rajarshi Ray, Olivier Lebeltel, Rodolfo Ripado, Antoine Girard, Thao Dang, and Oded Maler. Spaceex: Scalable verification of hybrid systems. In *Computer Aided Verification*, pages 379–395. Springer, 2011.

[For07] Otto Forster. *Integralrechnung im IRn mit Anwendungen*. Vieweg-Studium Aufbaukurs Mathematik. Vieweg, Wiesbaden, 2007.

[For11] Otto Forster. *Differentialrechnung im R n, gewöhnliche Differentialgleichungen*. Studium : Grundkurs Mathematik. 2011.

[FPNB08] Peter Fritzson, Adrian Pop, Kristoffer Norling, and Mikael Blom. Comment-and indentation preserving refactoring and unparsing for modelica. In *Proceedings of the 6th International Modelica Conference (Modelica 2008)*, pages 3–4, 2008.

[FTCW16] Simon Foster, Bernhard Thiele, Ana Cavalcanti, and Jim Woodcock. Towards a utp semantics for modelica. In *International Symposium on Unifying Theories of Programming*, pages 44–64. Springer, 2016.

[GDD+09] Dragan Ga, Dragan Djuric, Vladan Deved, et al. *Model driven engineering and ontology development*. Springer Science & Business Media, 2009.

[Gir05] Antoine Girard. Approximate bisimulations for constrained linear systems. *Proceedings of the 44th IEEE Conference on Decision and Control*, pages 1307–1317, 2005.

[GP05] Antoine Girard and George J. Pappas. Approximate bisimulations for nonlinear dynamical systems. In *Proceedings of the 44th IEEE Conference on Decision and Control*, pages 684–689. IEEE, 2005.

[GP07a] Antoine Girard and George J. Pappas. Approximate bisimulation relations for constrained linear systems. *Automatica*, 43(8):1307–1317, 2007.

[GP07b] Antoine Girard and George J. Pappas. Approximation metrics for discrete and continuous systems. *IEEE Transactions on Automatic Control*, 52(5):782–798, 2007.

[GP11] Antoine Girard and George J. Pappas. Approximate bisimulation: A bridge between computer science and control theory. *European Journal of Control*, 17(5):568–578, 2011.

[Gra12] Urs Graf. *Applied Laplace transforms and z-transforms for scientists and engineers: a computational approach using a Mathematica package*. Birkhäuser, 2012.

[GST12] Rafal Goebel, Ricardo G. Sanfelice, and Andrew R. Teel. *Hybrid Dynamical Systems: modeling, stability, and robustness*. Princeton University Press, 2012.

[HDE+08] Grégoire Hamon, Bruno Dutertre, Levent Erkok, John Matthews, Daniel Sheridan, David Cok, John Rushby, Peter Bokor, Sandeep Shukla, Andras Pataricza, et al. Simulink design verifier-applying automated formal methods to simulink and stateflow. In *AFM'08: Third Workshop on Automated Formal Methods 14 July 2008 Princeton, New Jersey*, 2008.

[Her04] Martin Hermann. *Numerik gewöhnlicher Differentialgleichungen: Anfangs- und Randwertprobleme*. Oldenbourg, München [u.a.], 2004.

[HHWT97] Thomas A Henzinger, Pei-Hsin Ho, and Howard Wong-Toi. Hytech: A model checker for hybrid systems. In *International Conference on Computer Aided Verification*, pages 460–463. Springer, 1997.

[HKAS14] Ashlie B Hocking, John Knight, M Anthony Aiello, and Shinichi Shiraishi. Proving model equivalence in model based design. In *Software Reliability Engineering Workshops (ISSREW), 2014 IEEE International Symposium on*, pages 18–21. IEEE, 2014.

[HKPV95] Thomas A Henzinger, Peter W Kopke, Anuj Puri, and Pravin Varaiya. What's decidable about hybrid automata? In *Proceedings of the twenty-seventh annual ACM symposium on Theory of computing*, pages 373–382. ACM, 1995.

[HMU01] John E Hopcroft, Rajeev Motwani, and Jeffrey D Ullman. Introduction to automata theory, languages, and computation. *ACM SIGACT News*, 32(1):60–65, 2001.

[HNNA09] Görel Hedin, Emma Nilsson-Nyman, and Johan Akesson. A plan for building renaming support for modelica. In *WRT'09: 3rd ACM Workshop on Refactoring Tools*, 2009.

[HRB13] Paula Herber, Robert Reicherdt, and Patrick Bittner. Bit-precise formal verification of discrete-time matlab/simulink models using smt solving. In *International Conference on Embedded Software*. IEEE Press, 2013.

[HWS+11] Wei Hu, Joachim Wegener, Ingo Stürmer, Robert Reicherdt, Elke Salecker, and Sabine Glesner. MeMo – methods of model quality. 2011.

[Int11] International Organization for Standardization. ISO 26262 - Road vehicles – Functional safety. (26262), 2011.

[JFA+07] A Agung Julius, Georgios E Fainekos, Madhukar Anand, Insup Lee, and George J Pappas. Robust test generation and coverage for hybrid systems. In *International Workshop on Hybrid Systems: Computation and Control (HSCC)*, volume 4416, pages 329–342. Springer, 2007.

[Kai80] Thomas Kailath. *Linear systems*, volume 156. Prentice-Hall Englewood Cliffs, NJ, 1980.

[KKR09] Matt Kaufmann, Jacob Kornerup, and Mark Reitblatt. Formal verification of labview programs using the acl2 theorem prover. In *Proceedings of the Eighth International Workshop on the ACL2 Theorem Prover and its Applications*, pages 82–89. ACM, 2009.

[KM97] Matt Kaufmann and J. Strother Moore. An industrial strength theorem prover for a logic based on common lisp. *IEEE Transactions on Software Engineering*, 23(4):203–213, 1997.

[KN12] Ulrich Krause and Tim Nesemann. *Differenzengleichungen und diskrete dynamische Systeme: eine Einführung in Theorie und Anwendungen*. Walter de Gruyter, 2012.

[Kå98] David Kågedal. A natural semantics specification for the equation-based modeling language modelica. *LiTH-IDA-Ex-98/48, Linköping University, Sweden*, 1998.

[Laz06] Mircea Lazar. *Model predictive control of hybrid systems: Stability and robustness*. Dissertation, Technische Universiteit Eindhoven, Eindhoven, 2006.

[LLQ+10] Jiang Liu, Jidong Lv, Zhao Quan, Naijun Zhan, Hengjun Zhao, Chaochen Zhou, and Liang Zou. A calculus for hybrid csp. In *Asian Symposium on Programming Languages and Systems*, pages 1–15. Springer, 2010.

[LT05] Ruggero Lanotte and Simone Tini. Taylor approximation for hybrid systems. In *Hybrid Systems: Computation and Control*, pages 402–416. Springer, 2005.

[LZ05] Edward A. Lee and Haiyang Zheng. Operational semantics of hybrid systems. In *Hybrid Systems: Computation and Control*, pages 25–53. Springer, 2005.

[Mat14] Stefan Matthew. Towards a taxonomy for simulink model mutations. In *IEEE Seventh International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pages 206–215. IEEE, 2014.

[MMBC11] Karthik Manamcheri, Sayan Mitra, Stanley Bak, and Marco Caccamo. A step towards verification and synthesis from simulink/stateflow models. In *Proceedings of the 14th international conference on Hybrid systems: computation and control*, pages 317–318. ACM, 2011.

[MMP91] Oded Maler, Zohar Manna, and Amir Pnueli. Prom timed to hybrid systems. In *Workshop/School/Symposium of the REX Project (Research and Education in Concurrent Systems)*, pages 447–484. Springer, 1991.

[MS98] Aloysius K Mok and Douglas Stuart. An rtl semantics for labview. In *Aerospace Conference*, volume 4, pages 61–71. IEEE, 1998.

[MSUY13] Rupak Majumdar, Indranil Saha, Koichi Ueda, and Hakan Yazarel. Compositional equivalence checking for models and code of control

systems. In *Decision and Control (CDC), 2013 IEEE 52nd Annual Conference on*, pages 1564–1571. IEEE, 2013.

[NH04] Jitse Niesen and Trinity Hall. *On the global error of discretization methods for ordinary differential equations*. PhD thesis, Citeseer, 2004.

[ORS92] Sam Owre, John M Rushby, and Natarajan Shankar. Pvs: A prototype verification system. In *International Conference on Automated Deduction*, pages 748–752. Springer, 1992.

[PGT08] Giordano Pola, Antoine Girard, and Paulo Tabuada. Approximately bisimilar symbolic models for nonlinear control systems. *Automatica*, 44(10):2508–2516, 2008.

[PL96] Stefan Pettersson and Bengt Lennartson. Stability and robustness for hybrid systems. In *Proceedings of the 35th IEEE Conference on Decision and Control*. IEEE, 1996.

[Pla08] André Platzer. Differential dynamic logic for hybrid systems. *Journal of Automated Reasoning*, 41(2):143–189, 2008.

[PP03] Stephen Prajna and Antonis Papachristodoulou. Analysis of switched and hybrid systems-beyond piecewise quadratic methods. In *American Control Conference, 2003. Proceedings of the 2003*, volume 4, pages 2779–2784. IEEE, 2003.

[PQ08] André Platzer and Jan-David Quesel. Keymaera: A hybrid theorem prover for hybrid systems (system description). In *Automated Reasoning*, pages 171–178. Springer, 2008.

[QML+15] Jan-David Quesel, Stefan Mitsch, Sarah Loos, Nikos Aréchiga, and André Platzer. How to model and prove hybrid systems with keymaera: a tutorial on safety. *International Journal on Software Tools for Technology Transfer*, pages 1–25, 2015.

[RG14] Robert Reicherdt and Sabine Glesner. Formal verification of discrete-time matlab/simulink models using boogie. In *Software Engineering and Formal Methods*, pages 190–204. Springer, 2014.

[RS09] Michael Ryabtsev and Ofer Strichman. Translation validation: From simulink to c. In *International Conference on Computer Aided Verification (CAV)*, volume 9, pages 696–701. Springer, 2009.

[Sal17] Ahmet Salman. Beweismethodik für Verhaltensäquivalenz von Simulink Modellen. Bachelor thesis, Technische Universitaet Berlin, 2017.

[SDS15] Muharrem Orkun Saglamdemir, Gunhan Dundar, and Alper Sen. An analog behavioral equivalence checking methodology for simulink models and circuit level designs. In *Synthesis, Modeling, Analysis and Simulation Methods and Applications to Circuit Design (SMACD), 2015 International Conference on*, pages 1–4. IEEE, 2015.

[Sel03] Bran Selic. The pragmatics of model-driven development. *IEEE software*, 20(5):19–25, 2003.

[SHGG15] Sebastian Schlesinger, Paula Herber, Thomas Göthel, and Sabine Glesner. Towards the verification of refactorings of hybrid simulink models. In Alexei Lisitsa, Andrei Nemytyk, and Alberto Pettorossi, editors, *Proceedings of the Third International Workshop on Verification and Program Transformation (VPT)*, volume 199 of *EPTCS*, page 69, 2015.

[SHGG16a] Sebastian Schlesinger, Paula Herber, Thomas Göthel, and Sabine Glesner. Proving correctness of refactorings for hybrid simulink models with control flow. In *Cyber Physical Systems. Design, Modeling, and Evaluation*, 2016.

[SHGG16b] Sebastian Schlesinger, Paula Herber, Thomas Göthel, and Sabine Glesner. Proving transformation correctness of refactorings for discrete and continuous simulink models. In *The Eleventh International Conference on Systems, EMBEDDED 2016, International Symposium on Advances in Embedded Systems and Applications*, pages 45–50, 2016.

[SHGG18] Sebastian Schlesinger, Paula Herber, Thomas Göthel, and Sabine Glesner. Verified model refactorings for hybrid control systems. In *PhD Forum Design Automation and Test in Europe (DATE)*, 2018.

[SK00a] B Izaias Silva and Bruce H Krogh. Formal verification of hybrid systems using checkmate: a case study. In *American Control Conference, 2000. Proceedings of the 2000*, volume 3, pages 1679–1683. IEEE, 2000.

[SK00b] B Izaias Silva and Bruce H Krogh. Modeling and verification of sampled-data hybrid systems. In *Proceedings of the 4th International Conference on Automation of Mixed Processes: Hybrid Dynamic Systems*, pages 237–242, 2000.

[Sko17] David Skowronek. Lösungsmethoden für Differenzengleichungen, Differentialgleichungen und retardierende Differentialgleichungen. Bachelor thesis, Technische Universitaet Berlin, 2017.

[SLZ08] Yang Yi Sui, Jun Lin, and Xiao Tuo Zhang. An automated refactoring tool for dataflow visual programming language. *ACM Sigplan Notices*, 43(4):21–28, 2008.

[SRKC00] B Izaias Silva, Keith Richeson, Bruce Krogh, and Alongkrit Chutinan. Modeling and verifying hybrid dynamic systems using checkmate. In *Proceedings of 4th International Conference on Automation of Mixed Processes*, volume 4, pages 1–7, 2000.

[SSD12] Muharrem Orkun Saglamdemir, Alper Sen, and Gunhan Dündar. A formal equivalence checking methodology for simulink and register transfer level designs. In *Synthesis, Modeling, Analysis and Simulation Methods and Applications to Circuit Design (SMACD), 2012 International Conference on*, pages 221–224. IEEE, 2012.

[SWP12] Karl Strehmel, Rüdiger Weiner, and Helmut Podhaisky. *Numerik gewöhnlicher Differentialgleichungen: nichtsteife, steife und differential-algebraische Gleichungen*. Springer Science & Business Media, 2012.

[Tau64] Selmo Tauber. Existence and uniqueness theorems for solutions of difference equations. *The American Mathematical Monthly*, 71(8):859–862, 1964.

[Tes12] Gerald Teschl. *Ordinary differential equations and dynamical systems*, volume 140 of *Graduate studies in mathematics*. American Mathematical Society, Providence, RI, 2012.

[The17] Inc. The Mathworks. Longitudinal flight control of grumman aerospace f-14, 2017.

[Tim08] Steffen Timmann. *Repetitorium Topologie und Funktionalanalysis*. Binomi Verlag, 2008.

[Tiw08] Ashish Tiwari. Abstractions for hybrid systems. *Formal Methods in System Design*, 32(1):57–83, 2008.

[TM17a] Inc. The Mathworks. *Simulink® Check User's Guide R2017b*. The Mathworks, Inc., 2017.

[TM17b] Inc. The Mathworks. *Simulink® User's Guide R2017b*. The Mathworks, Inc., 2017.

[TSCC05] Stavros Tripakis, Christos Sofronis, Paul Caspi, and Adrian Curic. Translating discrete-time simulink to lustre. *ACM Transactions on Embedded Computing Systems (TECS)*, 4(4):779–818, 2005.

[TWD13] Quang Minh Tran, Benjamin Wilmes, and Christian Dziobek. Refactoring of simulink diagrams via composition of transformation steps. In *The Eighth International Conference on Software Engineering Advances (ICSEA)*, pages 140–145, 2013.

[VCSS03] René Vidal, Alessandro Chiuso, Stefano Soatto, and Shankar Sastry. Observability of linear hybrid systems. In *International Workshop on Hybrid Systems: Computation and Control*, pages 526–539. Springer, 2003.

[WA10] Guoqiang Wang and Hugo A Andrade. Labview^{TM}: A graphical system design environment for adaptive hardware/software systems. In *NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, pages 82–82. IEEE, 2010.

[Wol15] Stephen Wolfram. *An elementary introduction to the Wolfram Language*. Wolfram Media, Incorporated, 2015.

[Wor16] Johannes Wortmann. Implementierung einer operationalen Semantik fuer Simulink. Bachelor thesis, Technische Universitaet Berlin, 2016.

Supervised Bachelor and Master Theses

[Bas16] Johannes Basler. Implementierung eines Checkers zum Nachweis der Korrektheit von Refactorings auf diskreten und kontinuierlichen Simulink Modellen. Bachelor thesis, Technische Universitaet Berlin, 2016.

[Ben15] Matthias Bentert. Implementierung von Transformationen von MATLAB/ Simulink Modellen. Bachelor thesis, Technische Universitaet Berlin, 2015.

[Dan16] Armin Danziger. Implementierung einer grafischen Lösung zur Darstellung von MATLAB / Simulink Modellen. Bachelor thesis, Technische Universitaet Berlin, 2016.

[Fro18] Dennis Froehlich. Eine semantisch korrekte Transformation von datenflussorientierten Blockdiagrammen zu Differential Dynamic Logic. Bachelor thesis, Technische Universitaet Berlin, 2018.

[Kul15] Gino Kulej. Entwicklung und Implementierung von Refactorings für MATLAB/ Simulink mittels eines Eclipse Modeling Frameworks. Bachelor thesis, Technische Universitaet Berlin, 2015.

[Sal17] Ahmet Salman. Beweismethodik für Verhaltensäquivalenz von Simulink Modellen. Bachelor thesis, Technische Universitaet Berlin, 2017.

[Sch17] Lukas Schaus. Automatic Assessment Generation fro Formalized Requirements for Model-Based Testing. Master thesis, Technische Universitaet Berlin, 2017.

[Sch18] Katja Schmidt. Generierung von Modellbausteinen zur Laufzeitverifikation von Simulink Modellen basierend auf formalen Anforderungen. Bachelor thesis, Technische Universitaet Berlin, 2018.

[Sko17] David Skowronek. Lösungsmethoden für Differenzengleichungen, Differentialgleichungen und retardierende Differentialgleichungen. Bachelor thesis, Technische Universitaet Berlin, 2017.

[Sta18] Georgios Stavrakakis. Information Flow Analysis for Simulink Models with Cyclic Control Signals. Master thesis, Technische Universitaet Berlin, 2018.

[Tri17] To Minh Triet. Eine Repräsentation von Ausdrücken, Differential- und Differenzengleichungen in MATLAB/ Simulink. Bachelor thesis, Technische Universitaet Berlin, 2017.

[Wor16] Johannes Wortmann. Implementierung einer operationalen Semantik fuer Simulink. Bachelor thesis, Technische Universitaet Berlin, 2016.

[Zbo16] Niklas Zbozinek. Implementierung einer Clone Detection fuer MATLAB/Simulink Modelle. Bachelor thesis, Technische Universitaet Berlin, 2016.

Publications by Sebastian Schlesinger

[DGL+15] Johannes Dyck, Holger Giese, Leen Lamber, Sebastian Schlesinger, and Sabine Glesner. Towards the automatic verification of behavior preservation at the transformation level for operational model transformations. In *Proceedings of the 4th Workshop on the Analysis of Model Transformations co-located with the 18th International Conference on Model Driven Engineering Languages and Systems (MODELS 2015)*, pages 36–45, 2015.

[SHGG15] Sebastian Schlesinger, Paula Herber, Thomas Göthel, and Sabine Glesner. Towards the verification of refactorings of hybrid simulink models. In Alexei Lisitsa, Andrei Nemytyk, and Alberto Pettorossi, editors, *Proceedings of the Third International Workshop on Verification and Program Transformation (VPT)*, volume 199 of *EPTCS*, page 69, 2015.

[SHGG16a] Sebastian Schlesinger, Paula Herber, Thomas Göthel, and Sabine Glesner. Proving correctness of refactorings for hybrid simulink models with control flow. In *Cyber Physical Systems. Design, Modeling, and Evaluation (CyPhy)*, pages 71–86, 2016.

[SHGG16b] Sebastian Schlesinger, Paula Herber, Thomas Göthel, and Sabine Glesner. Proving transformation correctness of refactorings for discrete and continuous simulink models. In *ICONS 2016, The Eleventh International Conference on Systems, EMBEDDED 2016, International Symposium on Advances in Embedded Systems and Applications*, pages 45–50, 2016.

[SHGG18a] Sebastian Schlesinger, Paula Herber, Thomas Göthel, and Sabine Glesner. Equivalence checking for hybrid control systems modelled in simulink. In *2018 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C)*, pages 572–579. IEEE, 2018.

[SHGG18b] Sebastian Schlesinger, Paula Herber, Thomas Göthel, and Sabine Glesner. Verified model refactorings for hybrid control systems. In *PhD Forum Design Automation and Test in Europe (DATE) 2018*, 2018.